GTI

Hannes Diener

ENC B-0123,
diener@math.uni-siegen.de

11.–18. Juli 2013

Komplexität

Einleitung

Im letzten Teil der Vorlesung soll es uns nicht so sehr darum gehen, ob ein Problem oder eine Funktion theoretisch berechenbar sind, sondern inwieweit dies auch praktisch vom Zeit- und Rechenaufwand machbar ist; d.h. wie *komplex* ein Problem ist.

Komplexität

Einleitung

Wie wir gesehen haben sind alle Rechenmodelle, die wir betrachtet haben gleich mächtig. Andererseits lassen sich einige Probleme z.B. viel einfacher mit `WHILE`-Programmen lösen als mit Turingmaschinen (man denke nur an die Addition einer Binärzahl mit 1). D.h. wie auch immer wir Komplexität definieren, der Begriff wird vom eigentlichen Modell und vielen anderen Details abhängig sein.

Komplexität

Einleitung

Wie wir gesehen haben sind alle Rechenmodelle, die wir betrachtet haben gleich mächtig. Andererseits lassen sich einige Probleme z.B. viel einfacher mit `WHILE`-Programmen lösen als mit Turingmaschinen (man denke nur an die Addition einer Binärzahl mit 1). D.h. wie auch immer wir Komplexität definieren, der Begriff wird vom eigentlichen Modell und vielen anderen Details abhängig sein. Allerdings werden wir auch sehen, daß für die meisten Komplexitätsklassen, die wir betrachten viele dieser Unterschiede wieder verschwinden.

Die Art von Komplexität eines Problems, die uns am meisten interessiert ist:

> *Wie viel Zeit benötigt ein Programm zur Berechnung einer Ausgabe?*

Es sei aber auch erwähnt, daß es auch Problemstellungen gibt bei denen man eher z.B. am Speicherbedarf, Bandbreitenbedarf, usw. interessiert ist.

Komplexität

Einleitung

Wir wollen uns im folgenden vor Allem auf den Zeitbedarf einer Turingmaschine konzentrieren. Auch wenn Turingmaschinen praktisch nicht relevant sind lassen sich die Ergebnisse dennoch in die Praxis übertragen. Ausserdem ist es anschaulicher den Zeitbedarf einer (Turing)-maschine zu definieren, als den von anderen Berechnungsmodellen.

Komplexität

Einleitung

Also kommen wir zu den ersten Definitionen.

Definition

Die Rechenzeit einer Turingmaschine M ist definiert durch

$$\mathrm{RZ}_M(x_1, \ldots, x_k) = \begin{cases} n & \text{falls } M \text{ in genau } n \text{ Rechenschritten} \\ & \text{bei Eingabe } x_1, \ldots, x_k \text{ akzeptiert} \\ \perp & \text{sonst.} \end{cases}$$

Definition

Eine Turingmaschine M berechnet die Funktion $f : \mathbb{N}^k \to \mathbb{N}$ in Zeit $t : \mathbb{N} \to \mathbb{N}$, falls für alle $x_1, \ldots, x_k \in \mathbb{N}$ gilt

$$\mathrm{RZ}_M(x_1, \ldots, x_k) \leqslant t(|x_1| + \cdots + |x_k|) \ ,$$

wobei $|x|$ die Länge der Binärzahldarstellung von x ist.

Es ist wichtig, sich daran zu erinnern, daß wir annehmen, daß Turingmaschinen auf Binärzahldarstellungen arbeiten.

Beispiel

Die Turingmaschine M_1, die 1 zu einer Binärzahl addiert benötigt:

$$\mathrm{RZ}_{M_1}(11) = 10 \ .$$

Allgemein sieht man, daß M_1 ca. $2|x|$ Schritte, bei Eingabe x benötigt.

Um uns nicht unnötig mit Details aufzuhalten (siehe “ca.” auf der vorherigen Folie) treffen wir noch folgende Konvention:

Definition

Eine Turingmaschine M berechnet f in Zeit $\mathcal{O}(t)$, falls es ein $c > 0$ gibt, so daß M die Funktion f in Zeit

$$ct(n) + c$$

berechnet.

Also berechnet M_1 die Funktion $S = \lambda n.n + 1$ in Zeit $\mathcal{O}(n)$.

Berechnet man das Laufzeitverhalten für verschiedene einfache Funktionen erhält man folgende Tabelle:[1]

		Eingabelänge				
		20	40	60	100	300
Laufzeitverhalten						
polynomiell	n	10^{-8} s	10^{-7} s	10^{-7} s	10^{-7} s	10^{-7} s
	n^2	10^{-7} s	10^{-6} s	10^{-6} s	10^{-5} s	10^{-4} s
	n^3	10^{-5} s	10^{-4} s	10^{-4} s	10^{-3} s	10^{-2} s
exponentiell	$2^{\frac{n}{\log_2 n}}$	10^{-7} s	10^{-6} s	10^{-5} s	10^{-3} s	79 Tage
	2^n	10^{-3} s	19 min	37 J.	10^{13} J.	10^{73} J.

[1] Die angegebenen Zeiten beziehen sich auf einen Rechner mit $1 Mio.$ Operationen pro Sekunde. Beachte, dass schnellere Rechner lediglich eine Beschleunigung um einen konstanten Faktor bewirken, d.h. das Bild nicht wesentlich ändern.

Wie man aus der Tabelle oben sieht sind Polynome praktisch berechenbar, aber exponentielle Funktionen nicht.

- Natürlich ist ein Algorithmus, der in Zeit n^2 arbeitet besser als einer, der Zeit n^5 benötigt. Für theoretische Untersuchungen braucht man aber zumindest, dass die betrachtete Funktionsklasse unter Substitution abgeschlossen ist, was bei Algorithmen mit polynomiellem Laufzeitverhalten der Fall ist, aber nicht bei solchen, die für festes k in Zeit n^k arbeiten.
- Natürlich ist auch z.B. 2^n für kleine n schneller berechenbar, als $n^{2000} + 1024$. Uns interessiert aber nur das asymptotische Verhalten.

Definition

Sei $f : \mathbb{N}^k \to \mathbb{N}$. Eine Turingmaschine M berechnet die Funktion f in Polynomialzeit, wenn es ein Polynom p gibt, so dass f von M in Zeit p berechnet wird. Sei

$$\mathbf{FP} = \{ f \mid f \text{ wird in Polynomialzeit berechnet} \}$$

$$\mathbf{P} = \left\{ A \subseteq \mathbb{N}^k \,\middle|\, \chi_A \in \mathbf{FP} \right\}$$

Beispiel

1. *Jedes Polynom ist in* **FP**
2. *Sei* $\exp(x) = 2^x$. *Dann ist* $\exp \notin$ **FP**, *da die Funktionswerte zu groß sind. Es ist* $|2^x| = x + 1 \geqslant 2^{|x|-1} + 1$. *Somit wächst die Länge des Funktionswerts exponentiell mit der Länge der Eingabe. Eine Turingmaschine benötigt also mindestens* $2^{|x|-1} + 1$ *Schritte, allein um den Funktionswert* 2^x *auf das Band zu schreiben.*
3. *Beachte aber, dass*

$$\left\{ (x, y, z) \in \mathbb{N}^3 \,\middle|\, x^y = z \right\} \in \mathbf{P}.$$

Lemma

1. *Die Klasse* **FP** *ist unter Komposition abgeschlossen.*
2. **P** *ist unter Vereinigung, Durchschnitt und Komplementbildung abgeschlossen.*
3. *Insbesondere ist wenn* $A \in \mathbf{P}$ *auch* $\overline{A} \in \mathbf{P}$.

Beweis.

Vorlesung. □

Beobachtung

Wie erwähnt ist jede Mehrbandturingmaschine durch eine Einbandmaschine in quadratischer Zeit simulierbar. Für **P** *und* **FP** *ist es also egal, welches Modell wir betrachten.*

Als Nächstes wollen wir uns der Komplexität entscheidbarer Probleme zuwenden. Wir werden insbesondere sehen, dass die möglicherweise große Laufzeit von Rechnungen nicht notwendigerweise durch einen langen Ausgabeprozess bedingt ist, sondern bei 0-1-wertigen Funktionen auftritt und also eine charakteristische Größe von Problemen ist, so wie es auch bei der Unentscheidbarkeit der Fall war.

Vorher wollen wir aber noch einige Beispiele von in Polynomzeit entscheidbaren Problemen auflisten. Sie treten alle in der Graphentheorie auf. Sei $G = (V, E)$ ein endlicher ungerichteter Graph. Ohne Einschränkung sei $V = \{1, \ldots, m\}$. Jeder endliche Graph ist vollständig durch seine Adjazenzmatrix

$$\begin{pmatrix} \delta_{11} & \ldots & \delta_{1m} \\ \vdots & \ddots & \vdots \\ \delta_{m1} & \ldots & \delta_{mm} \end{pmatrix} \quad \text{mit } \delta_{i,j} = \begin{cases} 1, & \text{falls } (i,j) \in E \\ 0, & \text{sonst.} \end{cases}$$

beschrieben. Diese Matrix kann als Wort

$$w_G = \delta_{11}\delta_{12}\ldots\delta_{1m}\delta_{21}\ldots\delta_{2m}\ldots\delta_{m1}\ldots\delta_{mm}$$

in eine Turingmaschine eingegeben werden.

Beispiel K_n...

Definition

1. k-FARB $= \{ G \mid G$ ist mit k Farben färbbar $\}$
2. ZH $= \{ G \mid G$ ist zusammenhängend $\}$
3. EK $= \{ G \mid G$ besitzt einen Eulerkreis $\}$

Lemma

Die Probleme 2-FARB, ZH *und* EK *sind in Zeit* $\mathcal{O}(n^2)$ *entscheidbar, also insbesondere in* $\mathbf{P}$.

Allerdings gibt es auch Probleme, die zwar ähnlich aussehen, aber für die kein Polynomzeitalgorithmus bekannt ist.

- Das Problem k-FARB, für $k > 2$.
- Das Problem HK $= \{ G \mid G \text{ besitzt Hamiltonkreis} \}$.
- Das Problem SOS (sum of subsets) mit

$$\text{SOS} = \{ (a_1, \ldots, a_m, b) \mid \exists c_1, \ldots, c_m \in \{0,1\} : (\textstyle\sum_{i=1}^{m} c_i a_i = b) \}$$

Allen betrachteten Problemen ist Struktur gemeinsam:

- $(\{1,\ldots,m\},K) \in k\text{-FARB} \iff$

$$(\exists (a_1,\ldots,a_m) \in \{1,\ldots,k\}^m) \\ \wedge\ (\forall i)(\forall j)(\{i,j\} \in K \implies a_i \neq a_j)$$

- $(\{1,\ldots,m\},K) \in \mathsf{HK} \iff$

$$(\exists (v_1,v_2,\ldots,v_m))\,\{v_1,\ldots,v_m\} = \{1,\ldots,m\} \\ \wedge\ (\forall 1 \leqslant i < m)\,\{v_i,v_{i+1}\} \in K \wedge \{v_m,v_1\} \in K$$

- $(a_1,\ldots,a_m,b) \in \mathsf{SOS} \iff$

$$(\exists c_1,\ldots,c_m \in \{0,1\}) \sum_{i=1}^{m} c_i a_i = b.$$

D.h. kennen wir eine Lösung, können wir diese einfach überprüfen.

Hierbei ist die Länge der Beschreibung der Lösung ein Polynom p in der Länge der Eingabe. In den obigen Beispielen kann $p(n) = n$ gewählt werden. Die Korrektheit der Lösung, d.h. die Gültigkeit der Forderung an die Lösung, kann in Polynomzeit überprüft werden. Wir wollen nun die Klasse **NP** der durch diese Eigenschaften charakterisierten Probleme näher untersuchen.

Definition (**NP**)

Eine Menge A ist in **NP**, *wenn es ein Polynom p und eine Menge $B \in \mathbf{P}$ gibt, so dass für alle $x \in \mathbb{N}$ gilt:*

$$x \in A \iff \exists y : [|y| \leqslant p(|x|) \wedge (x, y) \in B] \ .$$

Kurzer Exkurs: Dies ist vergleichbar mit folgender Charakterisierung von semi-entscheidbar und entscheidbar.

Satz

Eine Menge A ist genau dann rekursiv aufzählbar, wenn es eine entscheidbare Menge B gibt, so daß

$$x \in A \iff \exists y : \langle x, y \rangle \in B$$

Satz

Es gilt $\mathbf{P} \subseteq \mathbf{NP}$.

Beweis.

Sei $A \in \mathbf{P}$ beliebig. Wähle $p(n) = n$ und $B = A \times \{0\}$. Damit gilt:

$$\begin{aligned} x \in A &\iff (x, 0) \in B \\ &\iff \exists y : [|y| \leqslant |x| \wedge (x, y) \in B] \ . \end{aligned}$$

Also ist $A \in \mathbf{NP}$ □

Lemma

Für $k \geqslant 3$ *sind* k-FARB, HK, SOS $\in$ **NP**

Satz

Die Klasse **NP** *ist abgeschlossen unter Vereinigung und Durchschnitt, d.h., es gilt:*

$$A, B \in \mathbf{NP} \implies A \cup B, A \cap B \in \mathbf{NP}$$

Es ist nicht klar, ob **NP** unter Komplementen abgeschlossen ist.

Beobachtung

Eine große offene Frage ist

$$\mathbf{P} \stackrel{?}{=} \mathbf{NP}$$

Außer den oben aufgeführten Problemen sind noch zahlreiche Probleme aus so wichtigen Gebieten wie Kostenoptimierung, Transportplanung, Lagerhaltung und Stundenplanerstellung, um nur einige zu nennen, in **NP**. Für viele ist bisher nicht bekannt, ob sie in **P** sind.

Auf Grund der Definition der Klasse **NP** ist klar, dass man bei einem Problem $A \in \mathbf{NP}$ zur Entscheidung der Frage, ob $x \in A$ ist, die Menge der potentiellen Lösungen durchmustern kann, um festzustellen, ob es eine korrekte Lösung gibt. Für **NP**-Probleme, von denen wir nicht wissen, ob sie in **P** sind, ist kein anderes Verfahren bekannt, als das mehr oder weniger geschickte Durchmustern aller möglicher Lösungen. Davon gibt es aber $2^{p(n)}$ viele. Ist insbesondere $x \notin A$, so kann dies erst nach Durchlaufen dieses exponentiell großen Suchraumes festgestellt werden.

Definition

Sei **EXP** *die Gesamtheit der Mengen* $A \subseteq \mathbb{N}^m$ $m \geqslant 1$, *für die es ein Polynom p gibt, so dass A in Zeit* $2^{p(n)}$ *entschieden werden kann.*

Satz
Es gilt $\mathbf{NP} \subseteq \mathbf{EXP}$.

Ausserdem kann man zeigen, daß:

Satz
Es gilt $\mathbf{P} \subsetneq \mathbf{EXP}$.

Es ist aber nicht klar, ob

1. $\mathbf{P} = \mathbf{NP}$
2. $\mathbf{NP} = \mathbf{EXP}$
3. $\mathbf{P} \subsetneq \mathbf{NP} \subsetneq \mathbf{EXP}$.

Eine Umfrage unter 100 führenden Informatikern führte zu folgendem Ergebnis:

1. 61 denken $\mathbf{P} \neq \mathbf{NP}$.
2. 9 denken $\mathbf{P} = \mathbf{NP}$.
3. 4 denken es ist unabhängig von ZFC.
4. 3 sagten einfach nur nicht unabhängig von der Peano Arithmetik.
5. 5 sagten daß es vom Modell abhängt.
6. 22 hatten keine Meinung.

Und bis wann?

2002–2009	5
2010–2019	12
2020–2029	13
2030–2039	10
2040–2049	5
2050–2059	12
2060–2069	4
2070–2079	0
2080–2089	1
2090–2099	0
2100–2110	7
2110–2199	0
2200–3000	5

Im folgenden wollen wir noch eine alternative Charakterisierung von **NP** geben; via nichtdeterministischen Turingmaschinen. Nochmals zur Erinnerung: eine nichtdeterministische Turingmaschine ist aufgebaut wie eine normale Turingmaschine, allerdings haben wir keine Übergansfunktion, sondern eine Übergangsrelation. Somit gibt es zu einer Konfiguration oft mehrere Nachfolgekonfigurationen.

Wie wollen wir nun Zeitkomplexität ins Spiel bringen?

Wie wollen wir nun Zeitkomplexität ins Spiel bringen?

Definition

Eine nichtdeterministische Turingmaschine akzeptiert eine Menge $A^k \subset \mathbb{N}$ *in polynomieller Zeit* p*, falls jede akzeptierende Rechnung bei Eingabe* $\mathrm{bin}(x_1)\# \ldots \# \mathrm{bin}(x_k) \in A$ *höchstens* $p(|x_1| + \cdots + |x_k|)$ *Schritte benötigt.*

Satz

Die folgenden Aussagen sind äquivalent:

1. $A \in \mathbf{NP}$
2. *Es gibt eine nichtdeterministischen Turingmaschine N und ein Polynom p mit natürlichen Zahlen als Koeffizienten, so dass N die Menge A in Zeit p akzeptiert.*

Wenn man schon **P** = **NP**? nicht entscheiden kann, hat dann zumindest **NP** eine Struktur, die man untersuchen kann?

Definition (Polynomzeitreduzierbarkeit)

Seien $A, B \subseteq \mathbb{N}$.

1. A *ist polynomzeitreduzierbar auf* B, *wenn es eine Funktion* $f \in \mathbf{FP}$ *mit* $c_A = c_B \circ f$ *gibt. Wir schreiben dann* $A \leqslant^p B$.
2. $A \equiv^p B \iff A \leqslant^p B \wedge B \leqslant^p A$.

Beispiel: 3-FARB $\leqslant^p$ 4-FARB.

Lemma

Die Relation $\leqslant^p$ ist reflexiv und transitiv.

Ferner gilt:

Satz

1. $B \in \mathbf{P} \wedge A \leqslant^p B \implies A \in \mathbf{P}$
2. $B \in \mathbf{NP} \wedge A \leqslant^p B \implies A \in \mathbf{NP}$

Das nächste Resultat zeigt, dass die Mengen aus **P** die einfachsten Mengen in **NP** sind.

Satz

Seien $\varnothing, \mathbb{N} \neq B$ *und* $A \in \mathbf{P}$. *Dann ist* $A \leqslant^p B$.

Korollar

Sind A, B *nichtleere Mengen in* **P**, *die nicht schon* $\mathbb{N}$ *sind, so ist* $A \equiv_p B$.

Definition

Eine Menge B heißt **NP**-*vollständig*, wenn

- $B \in \mathbf{NP}$ ist und
- für alle $A \in \mathbf{NP}$ gilt, so dass $A \leqslant^p B$.

Oft spricht man von

Zunächst einmal wissen wir natürlich nicht, dass es solche Mengen überhaupt gibt. Das nächste Resultat zeigt u.a. die Bedeutung von **NP**-vollständigen Mengen bei der Lösung der Frage, ob $\mathbf{P} = \mathbf{NP}$.

Satz

*Sei B **NP**-vollständig. Dann gilt:*

1. $B \in \mathbf{P} \iff \mathbf{P} = \mathbf{NP}$
2. $(\forall C \in \mathbf{NP})\big[B \leqslant^p C \implies C \textit{ ist } \mathbf{NP}\textit{-vollständig}\big]$.

Die folgenden Probleme sind als **NP**-vollständig bekannt.

1. SAT
2. 3-SAT
3. Cliquen-Problem
4. Knotenüberdeckunsproblem
5. Hamiltonkreisproblem
6. Traveling Salesman

Wir zeigen **NP**-Vollständigkeit der Aussagen 1,3,4 indem wir

- zeigen daß SAT **NP**-vollständig ist,
- SAT $\leqslant^p$ Cliquen-Problem, und
- Cliquen-Problem $\leqslant^p$ Knotenüberdeckunsproblem.