Kapitel 4: Komplexitätstheorie

Gliederung

↓ mit minimalen Änderungen durch B. Baumgarten

Zielstellung

- Gegenstand der Betrachtung

  - In der Berechnungstheorie geht es darum, besser zu verstehen, dass es und warum es unlösbare algorithmische Probleme gibt.

  - In der Komplexitätstheorie geht es darum, besser zu verstehen, dass es und warum es lösbare algorithmische Probleme gibt, die sich nicht bzw. höchstwahrscheinlich nicht effizient lösen lassen

... also nur mit Computerprogrammen lösen lassen, die so viel Rechenzeit benötigen, dass man sie eigentlich nicht benutzen kann, die also nicht „praxistauglich“ sind

Illustration

- Warum ist die Ressource „Rechenzeit“ in der Praxis relevant?

  - Beim Public-Key-Verfahren RSA werden „große“ Primzahlen p und q verwendet, um den öffentlichen und den privaten Schlüssel für eine sichere (verschlüsselte) Kommunikation zu bestimmen
  - Um herauszubekommen, ob eine Zahl p eine Primzahl ist, könnte man sehr naiv wie folgt vorgehen:

    - Man testet der Reihe nach für alle n mit $2 \leq n < p$, ob n ein Teiler von p ist.
    - Wenn man einen Teiler findet, ist man fertig und weiß, dass p keine Primzahl ist.
    - Falls man darunter keinen Teiler von p gefunden hat, weiß man, dass p eine Primzahl ist.

Warum ist die Ressource „Rechenzeit“ in der Praxis relevant? (cont.)

- Um ein Gefühl dafür zu bekommen, ob dieser einfache Primzahltest praxistauglich ist, nehmen wir an, dass
  - man „sehr, sehr schnell“, sagen wir in 10^{-10} Sekunden, heraus-bekommt, ob ein Zahl n mit $2 \leq n < p$ ein Teiler von p ist

Größe von p	Rechenzeit
10^{10}	1 Sekunde
10^{20}	10^{10} Sekunden > 1 Jahr
10^{100}	10^{90} Sekunden > 10^{80} Jahre

Damit die Verschlüsselung mit RSA sicher ist, sollte man Primzahlen p und q benutzen, für die $p,q \geq 10^{100}$ gilt.

Anmerkung

- Auf die beschriebene Art werden die „großen“ Primzahlen, die man beim Public-Key-Verfahren RSA benötigt, natürlich nicht bestimmt.
  Da geht man anders vor, vgl. z.B. Texte über AKS-Primzahltestverfahren.

Zu klärende Begriffe

- Wir müssen uns auf darauf einigen, was hinter den folgenden Begriffen steckt
  - Rechenzeit eines Computerprogramms
  - „praxistaugliches“ Computerprogramm

Um diese Begriffe zu klären, diskutieren wir sie erst einmal mit Blick auf Programme des Berechnungsmodells Turing-Maschinen.

Begriff: Rechenzeit einer Turing-Maschine

- Sei Σ ein endliches Alphabet.
- Sei M eine Turing-Maschine mit dem Eingabealphabet Σ, die eine vollständig definierte Funktion über Σ^* berechnet.
- Sei $w \in \Sigma^*$.
- Dann definieren wir:

> Die Zahl $time_M(w)$ gibt an, wie viele Rechenschritte M benötigt, um die Eingabe zu verarbeiten, d.h. bis der Endzustand erreicht wird.

... d.h. wie viele Rechenschritte M benötigt, um die jeweilige Ausgabe zu berechnen

Ein Rechenschritt entspricht einem Konfigurationsübergang.

Beispiel

- Sei die folgende Turing-Maschine M mit dem Eingabealphabet $\Sigma = \{0,1\}$ gegeben

$\delta(q_0,B) = (q_e,0,N)$	$\delta(q_0,0) = (q_e,0,N)$	$\delta(q_0,1) = (q_1,1,R)$
$\delta(q_1,B) = (q_e,1,N)$	$\delta(q_1,0) = (q_e,0,N)$	$\delta(q_1,1) = (q_1,1,R)$

- Offenbar gilt:
  - $time_M(\varepsilon) = time_M(0) = time_M(00) = ... = 1$
  - $time_M(1) = time_M(10) = time_M(100) = ... = 2$
  - $time_M(11) = time_M(110) = time_M(1100) = ... = 3$
  - ...

Begriff: Obere Schranke für die Rechenzeit einer Turing-Maschine

- Sei Σ ein endliches Alphabet.
- Sei M eine Turing-Maschine mit dem Eingabealphabet Σ, die eine vollständig definierte Funktion über Σ^* berechnet.
- Sei s eine vollständig definierte Funktion über N_0.
- Dann definieren wir:

> Die Funktion s ist eine obere Schranke für die Rechenzeit der Turing-Maschine M, falls für alle $n \in N_0$ und für alle $w \in \Sigma^*$ gilt:
>
> - Wenn $|w| = n$ gilt, so gilt $time_M(w) \leq s(n)$.

Beispiel

- Sei die folgende Turing-Maschine M mit dem Eingabealphabet $\Sigma = \{0,1\}$ gegeben

$\delta(q_0,B) = (q_e,0,N)$	$\delta(q_0,0) = (q_e,0,N)$	$\delta(q_0,1) = (q_1,1,R)$
$\delta(q_1,B) = (q_e,1,N)$	$\delta(q_1,0) = (q_e,0,N)$	$\delta(q_1,1) = (q_1,1,R)$

- Offenbar gilt:
  - $time_M(w) \leq |w| + 1$
- Damit sind z.B. die Funktionen $s_1(n) = n+1$ und $s_2(n) = 2n$ obere Schranken für die Rechenzeit der Turing-Maschine M.

Begriff: „praxistaugliche“ Turing-Maschine

- Sei Σ ein endliches Alphabet.
- Sei M eine Turing-Maschine mit dem Eingabealphabet Σ, die eine vollständig definierte Funktion über Σ^* berechnet.
- Dann definieren wir:

> M ist eine polynomiell zeitbeschränkte Turing-Maschine, falls es ein $k \in N_0$ und ein $c \in N$ gibt, so dass das Polynom $p(n) := c \cdot n^k$ eine obere Schranke für die Rechenzeit der Turing-Maschine M ist.

Die Begriffe polynomiell zeitbeschränkte Turing-Maschine und „praxistaugliche“ Turing-Maschine sind als Synonyme zu verstehen.

- Diskussion des Begriffs „praxistaugliche“ Turing-Maschine

  - Die folgenden Tabellen illustrieren, weshalb der Begriff „praxistaugliche“ Turing-Maschine in der angegebenen Art präzisiert wird

  - Dazu seien Turing-Maschinen M_1, ..., M_5 gegeben,

    - die dieselbe Funktion berechnen und
    - deren Rechenzeit sich unterscheidet. (Die angegebenen oberen Schranken s_1, ..., s_5 seien „sinnvoll“ gewählt.)

Diskussion des Begriffs „praxistaugliche“ Turing-Maschine (cont.)

- Unter der Annahme, dass eine Turing-Maschine je Sekunde ca. 10^6 Rechenschritte ausführen kann, benötigen M_1, ..., M_5 die angegebene Rechenzeit, um Eingaben der Größe n zu verarbeiten

	n = 10	n = 20	n = 30	n = 40
M_1 mit $s_1(n) = n$	0,00001 s	0,00002 s	0,00003 s	0,00004 s
M_2 mit $s_1(n) = n^2$	0,0001 s	0,0004 s	0,0009 s	0,0016 s
M_3 mit $s_3(n) = n^3$	0,001 s	0,008 s	0,027 s	0,064 s
M_4 mit $s_4(n) = 2^n$	0,001 s	1,0 s	17,9 min	12,7 Tage
M_5 mit $s_5(n) = 3^n$	0,059 s	58 min	6,5 Jahre	3855 Jahr-hunderte

Diskussion des Begriffs „praxistaugliche“ Turing-Maschine (cont.)

- Wenn man davon ausgeht, dass man tolerieren kann, dass eine Turing-Maschine eine Stunde Zeit hat, um ihre Ausgabe zu bestimmen, kann man mit M_1, ..., M_5 Eingaben der folgenden Größe verarbeiten

	aktuell	mit 100-mal schnelleren Prozessoren	mit 1000-mal schnelleren Prozessoren
M_1 mit $s_1(n) = n$	N_1	100 N_1	1000 N_1
M_2 mit $s_2(n) = n^2$	N_2	10 N_2	31,6 N_2
M_3 mit $s_3(n) = n^3$	N_3	4,6 N_3	10 N_3
M_4 mit $s_4(n) = 2^n$	N_5	N_5 + 6,6	N_5 + 9,9
M_5 mit $s_5(n) = 3^n$	N_6	N_6 + 4,2	N_6 + 6,3

Zwischenbilanz

- Es gibt „pragmatische“ Gründe, den Begriff „praxistaugliche“ Turing-Maschine in der angegebenen Art zu präzisieren.
- Aber eigentlich benutzen wir z.B. Java- oder C++-Programme anstelle von Programmen des Berechnungsmodells Turing-Maschinen.

Das ist de facto kein Problem, weil …

Diskussion

- Anmerkungen

  - Man kann die beiden zentralen Begriffe Rechenzeit eines Programms sowie „praxistaugliches“ Programm völlig analog für Programme jedes anderen Berechnungsmodells definieren.

  - Dazu genügt es, zu präzisieren, was ein „Rechenschritt“ im jeweils betrachteten Berechnungsmodell ist

Üblicherweise wird jede Ausführung einer Elementaroperation (im zugrunde liegenden Verarbeitungsmodell) als Rechenschritt verstanden

Ein zentraler Zusammenhang

- Sei Σ ein endliches Alphabet
- Sei f eine vollständig definierte Funktion über Σ^*
- Dann gilt:

> Wenn es ein „praxistaugliches“ Java- oder C++-Programm gibt, mit dem man die Funktion f berechnen kann, dann gibt es auch eine „praxistaugliche“ Turing-Maschine, die die Funktion f berechnet.

Zu jedem Java- oder C++-Programm P gibt es eine äquivalente Turing-Maschine M.

P und M unterscheiden sich mit Blick auf ihre Rechenzeit nur „unwesentlich“.

Anmerkung

- Der auf der letzten Folie formulierte Zusammenhang, die so genannte erweiterte Churchsche These, gilt mit Blick auf fast alle bekannten Berechnungsmodelle
  - Zu diesen Berechnungsmodellen gehören alle, für die die uns bekannten „Hardware“ ein geeignetes Verarbeitungsmodell ist.
  - Ausnahmen sind die Berechnungsmodelle DNA-Computer und Quantencomputer, die eine andere Art „Hardware“ verwenden.