Komplexitätstheorie

Organisatorisches

- Zeit und Ort:

  Di 18-20 MZH 1470 und Do 18-20 MZH 1460

- Vortragende:

  Prof. Carsten Lutz
  Cartesium 2.59
  Tel. (218)-64431
  clu@uni-bremen.de

  Dr. Thomas Schneider
  Cartesium 2.56
  Tel. (218)-64432
  tschneider@informatik.uni-bremen.de

- Position im Curriculum:

  Wahlbereich Master-Ergänzung, Modulbereich Theorie, Profile SQ + KIKR

Organisatorisches

- Voraussetzungen:
  Grundvorlesung Theoretische Informatik

- Form: K4, voraussichtlich 6 Termine mit Übungen

- Vorlesungsmaterial:

  Folien und Aufgabenblätter auf:

  http://www.informatik.uni-bremen.de/tdki/lehre/ws14/kt/

  Beispiele, Beweise, etc an der Tafel (mitschreiben!)

Literatur

- Sanjeev Arora, Boaz Barak. Computational Complexity: A Modern Approach. Cambridge University Press, 2009.
- Oded Goldreich. Computational Complexity: a Conceptual Perspective. Cambridge University Press, 2008.
- Christos H. Papadimitriou. Computational Complexity. Addison-Wesley, 1994.
- Ingo Wegener. Komplexitätstheorie - Grenzen der Effizienz von Algorithmen. Springer, 2003.
- Dexter Kozen. Theory of Computation. Springer, 2006.
- The Parallel Computation Project: http://cs.armstrong.edu/greenlaw/research/PARALLEL/index.html

Prüfungen

Übungen:

- Übungsaufgaben jede zweite Woche (mit Zusatzaufgaben)
- Werden in Gruppen (2-3 Personen) bearbeitet, abgegeben und korrigiert
- Jeder Teilnehmer muss mindestens eine Aufgabe in der Übungsgruppe vorrechnen

oder

Mündliche Prüfung

Komplexitätstheorie

Ziel der Komplexitätstheorie

Analyse der inhärenten Komplexität von Problemen:

- Komplexität: hauptsächlich Laufzeit, aber auch andere Ressourcen (insbesondere Platzbedarf)
- Inhärent: Laufzeit / Ressourcenverbrauch des bestmöglichen Algorithmus für gegebenes Problem.
- Zwei Sichtweisen:

  Problemzentriert:
  Wie ressourcenaufwändig ist gegebenes Problem?

  Ressourcenzentriert:
  Welche Probleme kann ich mit gegebenen Ressourcen lösen

  Also: was ist Grenze der Berechenbarkeit unter beschränkten Ressourcen?

Beispiel: Erreichbarkeit in Graphen

Definition Pfad, Erreichbar

Sei $G = (V, E)$ gerichteter Graph. *Pfad* in G ist Folge von Knoten $v_0, \ldots, v_n$ so dass $(v_i, v_{i+1}) \in E$ für alle $i < n$.
Knoten v' *erreichbar* von Knoten v in G wenn es Pfad $v_0, \ldots, v_n$ gibt mit $v = v_0$ und $v_n = v'$.

Erreichbarkeitsproblem:

gegeben G, v, v', entscheide ob v' erreichbar von v in G.

Zentrales Problem der Informatik, z.B. Erreichbarkeit von Knoten in Netzwerken, analyse von Automaten, und vieles andere mehr.

Beispiel: Erreichbarkeit in Graphen

Algorithmus 1

Zähle alle Folgen von Knoten $v_0, \ldots, v_n$ auf mit $n \leq |V|$.
Überprüfe für jede Folge, ob

- $v = v_0$ und $v_n = v'$
- $(v_i, v_{i+1}) \in E$ für alle $i < n$.

Antworte “erreichbar”, wenn dies für mind. eine Folge zutrifft.
Sonst antworte “unerreichbar”.

Korrektheit:

- Wenn Algorithmus “erreichbar” antwortet, so ist dies offens. korrekt;
- Wenn v' von v erreichbar, so gibt es Pfad von v zu v', auf dem kein Knoten doppelt vorkommt
  Dieser Pfad hat max. Länge $|V|$ und wird vom Algorithmus gefunden.

Beispiel: Erreichbarkeit in Graphen

Algorithmus 1

Zähle alle Folgen von Knoten $v_0, \ldots, v_n$ auf mit $n \leq |V|$.
Überprüfe für jede Folge, ob

- $v = v_0$ und $v_n = v'$
- $(v_i, v_{i+1}) \in E$ für alle $i < n$.

Antworte “erreichbar”, wenn dies für mind. eine Folge zutrifft.
Sonst antworte “unerreichbar”.

Für jeden Pfad kann die angegebene Bedingung in Zeit $p(n)$ überprüft werden, mit $p(\cdot)$ Polynom

Wenn $|V| = n > 1$, so untersucht der Algorithmus mehr als $n^n \geq 2^n$ Pfade

Insgesamt braucht der Algorithmus also exponentiell viel Zeit

Beispiel: Erreichbarkeit in Graphen

Natürliche Frage:

Ist das Problem inhärent schwer oder der Algorithmus schlecht?

Algorithmus 2

$S := \{v\}$
markiere v
while $S \neq \emptyset$ do
    wähle $u \in S$
    $S := S \setminus \{u\}$
    for all $(u, u') \in E$ do
        if u' nicht markiert then
            $S := S \cup \{u'\}$
            markiere u'
        endif
    endfor
endwhile
wenn v' markiert antworte “erreichbar”, sonst “unerreichbar”

Beispiel: Erreichbarkeit in Graphen

Es ist nicht schwer, zu zeigen, dass Algorithmus 2 korrekt ist (Übung)

Laufzeitanalyse:

- Jeder Knoten wird höchstens einmal zu S hinzugefügt (beim Markieren)
- Die while Schleife macht also höchstens $n = |V|$ Schritte.
- Da jeder Knoten höchstens n Nachbarn hat, macht die “for all” Schleife höchstens n Schritte
- Insgesamt $\mathcal{O}(n^2)$ Schritte: polynomiell!

Algorithmus 1 war also tatsächlich suboptimal!!

Weiter optimierter Algorithmus hat sogar lineare Laufzeit ($\mathcal{O}(n)$ Schritte)

Beispiel: Cliquenproblem

Definition Clique

Sei $G = (V, E)$ ein ungerichteter Graph. *Clique* in G ist nicht-leere Knotenmenge $C \subseteq V$ so dass $\{v, v'\} \in E$ für alle $v, v' \in C$.
Die *Größe* einer Clique ist die Anzahl der darin enthaltenen Knoten.

Cliquenproblem:

gegeben G, Zahl k, entscheide ob G Clique der Größe k hat.

Beispiel: Cliquenproblem

Algorithmus

Zähle alle Teilmengen $C \subseteq V$ der Kardinalität k auf.
Für jede solche Teilmenge prüfe, ob sie Clique ist.
Wenn eine Clique gefunden wurde, antworte “ja”
Sonst antworte “nein”.

Der Algorithmus ist offensichtlich korrekt.

Laufzeit:

- Es kann in polynomieller Zeit geprüft werden, ob gegebenes $C \subseteq V$ eine k-Clique ist.
- Wenn $|V| = n$, dann gibt es $\binom{n}{k}$ Teilmengen der Größe k;
- Wenn z.B. $k = n/2$, dann $\binom{n}{k} \geq 2^n$, also exponentieller Zeitbedarf!

Beispiel: Cliquenproblem

Wieder die Frage:

Ist das Problem inhärent schwer oder der Algorithmus schlecht?

Beachte: der Algorithmus hat viel Ähnlichkeit mit Algorithmus 1 für Erreichbarkeit!

Antwort:

1. Das ist unbekannt (wie für viele natürliche Probleme)
2. Die Komplexitätstheorie erlaubt es uns trotzdem, das Cliquen-problem als "schwierig" zu identifizieren (NP-Vollständigkeit)

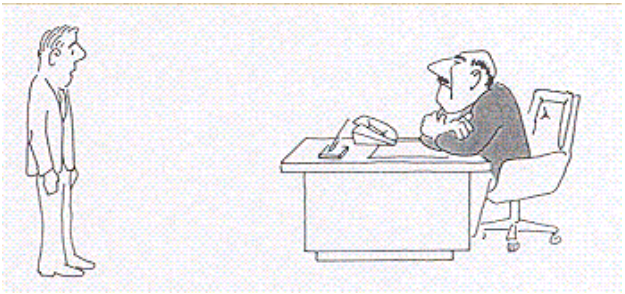
Wir brauchen keine Zeit damit zu verschwenden, nach einem polynomiellen Algorithmus zu suchen!

Beispiel: Cliquenproblem

Diese Antwort ist typisch für die Komplexitätstheorie:

- es gibt in diesem Bereich viele schwierige und ungelöste Probleme, darunter die bekanntesten in der Informatik
- nur wenig Resultate über die nicht-Machbarkeit (z.B. "das Cliquenproblem kann nicht in polynomieller Zeit gelöst werden")
- Stattdessen relative Aussagen:
  "Problem X ist mindestens so schwer wie all diese anderen Probleme hier, für keines ist effizienter Algorithmus bekannt"
- Auf diese Weise ergibt sich eine reiche Struktur im Raum aller Probleme
- Dies gibt einem ein sehr gutes Werkzeug in die Hand, um die Schwierigkeit eines gegebenen Problems einzuschätzen (sehr wichtig auch in der Praxis!)

Klassiker



Ich konnte keinen effizienten Algorithmus finden, ich bin zu blöd

Ich konnte keinen effizienten Algorithmus finden, weil es keinen gibt

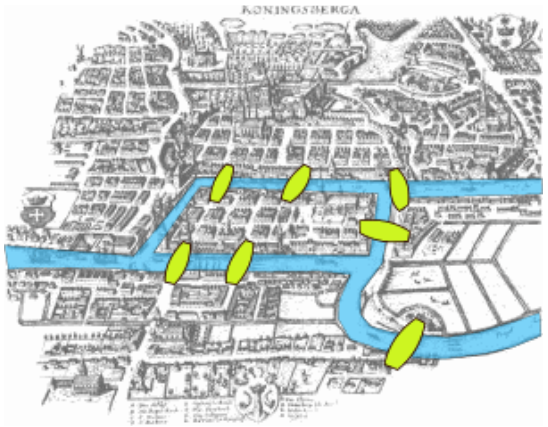
Gäbe es einen effizienten Algorithmus, dann auch für tausende andere Probleme, an denen sich Generationen von Forschern die Zähne ausgebissen haben.

Hätten Sie's geahnt?

Manchmal entscheiden Kleinigkeiten über die Schwierigkeit eines Problems:

Einfach (polynomielle Zeit):

Euler-Kreis: gegeben ungerichteter Graph, entscheide ob es einen Pfad gibt, der jede Kante genau einmal besucht und wieder am Anfang ankommt.



Schwer (NP-vollständig):

Hamilton-Kreis: gegeben ungerichteter Graph, entscheide ob es einen Pfad gibt, der jeden Knoten genau einmal besucht und wieder am Anfang ankommt.

Hätten Sie's geahnt?

Manchmal entscheiden Kleinigkeiten über die Schwierigkeit eines Problems:

Einfach (polynomielle Zeit):

Minimaler Schnitt: gegeben ungerichteter Graph $G = (V, E)$ und Zahl k, kann man V in Mengen V_1, V_2 partitionieren, so dass es höchstens k Kanten zwischen Elementen von V_1 und V_2 gibt?

Schwer (NP-vollständig):

Maximaler Schnitt: gegeben ungerichteter Graph $G = (V, E)$ und Zahl k, kann man V in Mengen V_1, V_2 partitionieren, so dass es mindestens k Kanten zwischen Elementen von V_1 und V_2 gibt?

Übersicht Vorlesung

- Kapitel 1: Einführung
- Kapitel 2: Turingmaschinen
- Kapitel 3: P vs. NP
- Kapitel 4: Mehr Ressourcen, mehr Möglichkeiten?
- Kapitel 5: Platzkomplexität
- Kapitel 6: Schaltkreise
- Kapitel 7: Orakel

Kapitel 1

Vier Klarstellungen:

- Laufzeitanalyse
- Worst Case Komplexität
- Repräsentation der Eingabe
- Entscheidungsprobleme vs. Berechnungsprobleme vs Optimierungsprobleme

Laufzeitanalyse

Laufzeit ist Funktion:
Anzahl Schritte in Abhängigkeit von Größe der Eingabe

Beim Beschreiben der Funktion abstrahieren wir häufig stark:

- Konstanten werden ignoriert ($\mathcal{O}$-notation)
- Meist ignorieren wir sogar den Grad von Polynomen

Insbesondere interessieren wir uns für polynomielle vs. exponentielle Laufzeit

Eingabegröße	1	2	3	4	5	6	7	8		20	128
n^2	1	4	9	16	25	36	49	64		400	16,384
2^n	2	4	8	16	32	64	128	256		1.048.576	AUTSCH!

"AUTSCH": Moderner Prozessor braucht mehr Zeit als vom Anfang des Universums bis heute

Worst Case Komplexität

Welche Art von Komplexität studieren wir?

Laufzeit

- ist obere Schranke für alle Eingaben derselben Größe
- bezieht sich also auf den schlimmstmöglichen Fall

Man spricht von Worst Case Komplexität

Worst Case Komplexität

Auf bestimmten Eingaben/Eingabeklassen kann der Algorithmus aber viel besser sein als im Worst Case

Beispiel Cliquenproblem:

Betrachte Klasse derjenigen Graphen, in denen vielen Knoten nur sehr wenig Nachbarn haben.

Dann ist folgende Heuristik sehr effektiv:

vor dem Suchen einer Clique der Größe k, eliminiere alle Knoten mit weniger als k Nachbarn

Die Worst-Case Komplexität des naiven Algorithmus ändert sich dadurch jedoch überhaupt nicht!

Worst Case Komplexität

Auch möglich ist die Untersuchung der Laufzeit auf typischen Eingaben (average case Komplexität).

Dies ist jedoch i.d.R. schwieriger:

- man braucht eine gute Charakterisierung von typischen Eingaben (oft schwer zu bestimmen)
- typische Eingaben werden oft über Wahrscheinlichkeitsverteilungen beschrieben (technisch anspruchsvoll)

In der klassischen Komplexitätstheorie und dieser VL geht es:

- um worst case Komplexität (vorsichtiger Ansatz!)
- nicht um Heuristiken / average case Komplexität

Repräsentation der Eingabe

Eingabe kann i.d.R. verschieden repräsentiert werden:

- Graphen: Adjazenzmatrix oder Liste von Kanten
- Zahlen: z.B. unär/binär/dezimal kodiert.

Größe kann von Repräsentation abhängen:

- Zahl n: unär Größe n, binär Größe $\log(n)$, jede andere Basis $\frac{1}{c} \cdot \log(n)$
- Kantenloser Graph mit n Knoten: Adjazenzmatrix hat immer Größe n^2, Größe der Kantenliste hängt von Anzahl Kanten ab

Repräsentation der Eingabe

Also hängt auch die Laufzeit von Repräsentation ab, z.B.:

- Man kann einfach in polynomieller Zeit entscheiden, ob Zahl prim ist, wenn diese unär kodiert ist.
- Für binäre Kodierung konnte dies erst 2002 von Agrawal, Kayal und Saxena gezeigt werden (“AKS Primality Test”).

Meist unterscheidet sich Größe versch. Repräsentationen nur polynomiell, z.B. Adjazenzmatrix vs. Kantenliste

Daher werden wir die Repräsentation meist nicht im Detail fixieren

Ausnahme: Zahlen (wir nehmen stets binäre Kodierung an!)

Kapitel 1

Entscheidungs- / Berechnungs- / Optimierungsprobleme

Varianten algorithmischer Problem

Algorithmische Probleme haben meist (mindestens) drei Varianten.

Am Beispiel des Cliquenproblems:

(a) Entscheidungsproblem (wie vorher)
Gegeben G und k, entscheide ob G eine k-Clique hat

(b) Berechnungsproblem
Gegeben G und k, berechne eine k-Clique in G
(gib $\perp$ aus, wenn keine existiert)

(c) Optimierungsproblem
Gegeben G, berechne eine Clique in G von maximaler Größe

Welche Variante die natürlichste ist, hängt von Problem und Anwendung ab.

Varianten algorithmischer Probleme

Komplexitätstheorie konzentriert sich auf Entscheidungsprobleme:

- einfacher zu handhaben (z.B. keine Optimierungsfunktion)
- für "natürliche" Probleme sind die Ressourcen, die für Entscheidungs- / Berechnungs- / Optimierungsproblem benötigt werden, sehr ähnlich.

Beispiel: polynomielle Algorithmen für das Cliquenproblem.

1. Optimierungsproblem polynomiell => Entscheidungsproblem polynomiell

   Eingabe Entscheidungsproblem: Graph G, Cliquengröße k

   Algorithmus für Optimierungsproblem liefert in Polyzeit größte Clique C

   G hat k-Clique gdw. $|C| \geq k$

Varianten algorithmischer Probleme

2. Entscheidungsproblem polynomiell => Berechnungsproblem polynomiell

Eingabe Berechnungsproblem: Graph G, Cliquengröße k

Zunächst entscheiden wir in Polyzeit, ob G eine k-Clique enthält
und geben $\perp$ aus, wenn das nicht der Fall ist.

procedure b-clique(G, k)
  $G' = G$
  while G' enthält k-Clique do
    wähle Knoten $v \in V$
    entferne v aus G'
  done
  gib v aus (letzter entfernter Knoten ist Teil der berechneten Clique)
  Entferne alle Knoten aus G', die in G nicht adjazent zu v sind
  if $k = 1$ then stop
  b-clique$(G', k - 1)$

Varianten algorithmischer Probleme

3. Berechnungsproblem polynomiell => Optimierungsproblem polynomiell

Eingabe Optimierungsproblem: Graph G

Berechne Cliquen für alle $k = 1, \ldots, |V|$.
Sobald $\perp$ zurückgegeben wird, wurde max. Clique gefunden

Polynomiell, da nur polynomiell viele Aufrufe des polynomiellen Berechnungsproblems notwendig sind.

Für andere Probleme ist hier oft binäre Suche erforderlich.
(wenn eine binär kodierte Zahl die Eingabe ist, deren Wert nicht von einer anderen Eingabe dominiert wird)

Übersicht Vorlesung

- Kapitel 1: Einführung
- Kapitel 2: Turingmaschinen
- Kapitel 3: P vs. NP
- Kapitel 4: Mehr Ressourcen, mehr Möglichkeiten?
- Kapitel 5: Platzkomplexität
- Kapitel 6: Schaltkreise
- Kapitel 7: Orakel