Zeitkomplexität (1)

Proseminar Theoretische Informatik

Warum Komplexitätsbetrachtung?

- Ein im Prinzip entscheidbares und berechenbares Problem kann in der Praxis trotzdem nicht (effizient) lösbar sein.
  - benötigt unverhältnismäßige Zeit
  - oder inakzeptablen Speicheraufwand
- Man versucht algorithmische Probleme (formalisiert als Sprachen) nach dem Bedarf an Berechnungsressourcen bei ihrer Lösung zu klassifizieren.

Zeitkomplexität

- Was meint Zeitkomplexität?
  - Anzahl der Rechenschritte eines Algorithmus zur Lösung eines Problems in Abhängigkeit von Länge der Eingabe
  - Grundlage für Betrachtung ist formales Maschinenmodell, meist die Turingmaschine
  - deterministische und nichtdeterministische Maschinen werden unterschieden, obwohl sie gleiche Sprachklasse erkennen

Laufzeitbestimmung

- Zeitkomplexität einer Turingmaschine *T* ist die maximale Anzahl von Schritten, die *T* für die Berechnung des Entscheidungsproblem eines Eingabewortes oder zur Berechnung einer Funktion benötigt.
  - *worst case* Betrachtung
  - Komplexität in O-Notation, asymptotische Abschätzung der Laufzeit, keine genaue Bestimmung

Ein Beispiel

- Geg. folgende Sprache: $A=\{0^k1^k|k\geq 0\}$
  - A ist offensichtlich entscheidbar, da kontextfrei
- Wie lange braucht eine deterministische Einband-Turingmaschine M_1, um A zu entscheiden?
  - Betrachtung der maximalen Anzahl von Schritten
- Laufzeit ist Funktion $f : \mathbb{N} \rightarrow \mathbb{N}$, f(n) sei Maximum der Schritte für alle Eingaben der Länge n

Ein Beispiel

M_1 funktioniert folgendermaßen:

1. Laufe das Band entlang und halte nicht akzeptierend, wenn rechts von einer 1 eine 0 gefunden wird
2. Wiederhole nächsten Schritt solange noch sowohl Nullen als auch Einsen auf dem Band sind
3. Laufe über das Band und überschreibe eine 0 und eine 1 mit einem Sonderzeichen
4. Stehen nur noch Sonderzeichen auf dem Band – akzeptiere, sonst – halte nicht akzeptierend

Ein Beispiel - Analyse

- Jede Stufe einzeln betrachten
  - Stufe 1: n Schritte hin, n Schritte zurück → O(n)
  - Stufe 2 und 3: Jeder Durchlauf dauert O(n) Schritte, immer zwei Ziffern werden überschrieben, also höchstens $^{n}/_{2}$ Durchläufe
    → $(n/2)O(n) = O(n^2)$
  - Stufe 4: einmal über das Band laufen → O(n)
- Gesamt: $O(n)+O(n^2)+O(n) = O(n^2)$

Zeitkomplexitätsklassen

- Definition:

  Sei t : $\mathbb{N} \rightarrow \mathbb{R}^+$, dann ist **TIME(t(n))** die Klasse aller Sprachen, die durch eine O(t(n))-zeitbeschränkte Turingmaschine entscheidbar sind.

- Für das vorherige Beispiel gilt: $A \in \text{TIME}(n^2)$
- sehr allgemein, etwas spezieller:
  - DTIME(t(n)), NTIME(t(n)), EXPTIME, NEXPTIME

Verbesserung von M_1

- Eine Mehrband-Turingmaschine entscheidet A schneller, nämlich in O(n) Zeit
- M_2 sei eine TM mit zwei Bändern:
  1. Für jede gelesene 0 auf Band 1 wird eine 0 auf das zweite Band kopiert
  2. Für jede 1 auf Band 1 wird dann eine 0 auf dem zweiten Band mit einem Sonderzeichen überschrieben
  3. Wenn alle Nullen überschrieben und alle Einsen gelesen sind – halte akzeptierend

Folgerung

- Die Komplexität von A hängt offensichtlich vom gewählten Maschinenmodell ab.
- Unterschied zwischen Berechenbarkeitstheorie und Komplexitätstheorie
  - entscheidbare Sprachen sind nicht alle in der selben Komplexitätsklasse, sondern dies ist abhängig von der Wahl des Modells

Church-Turing-These

- *Die Klasse der Turing-berechenbaren Funktionen entspricht genau der Klasse der intuitiv berechenbaren Funktionen.*

  - Impliziert: alle Berechenbarkeitsmodelle sind äquivalent
  - Hinsichtlich ihres Zeitbedarfs sind sie aber nicht äquivalent.

Vergleich Einband-, Mehrband-TM

- Einband-Turingmaschine „verschwendet“ Rechenzeit durch ständiges hin und her Bewegen des Kopfes auf dem Band
  - Schritte sind nicht direkt an der Lösung des Problems beteiligt

Satz: Sei $t(n)$ eine Funktion mit $t(n) \geq n$. Dann gibt es für jede $t(n)$-zeitbeschränkte Mehrband-Turingmaschine eine äquivalente $O(t^2(n))$-zeitbeschränkte Einband-Turingmaschine.

Simulation einer Mehrband-TM

- Beweis: Analyse der zusätzlichen Zeit
  - Gegeben M_1 und M_k
  - Annahme: M_k läuft in $t(n) \geq n$ Zeit
- M_1 simuliert einen Schritt von M_k folgendermaßen
  1. Aufsammeln der Informationen: Kopfposition und Zeichen
  2. Simulation eines Übergangs: Aktualisierung der Zeichen und der Kopfpositionen

Simulation einer Mehrband-TM

- Analyse
  - M_k benötigt maximal t(n) Schritte, um über ihre k Bänder zu laufen, M_1 braucht daher auch nicht länger
    - O(t(n)) ist obere Schranke für das Überstreichen des Bandes
  - M_1 simuliert jeden der t(n) Schritte von M_k durch O(t(n)) Schritte und braucht damit

$$t(n) \cdot O(t(n)) = O(t^2(n))$$

q.e.d

- Klassifizierung unabhängig von bestimmtem Maschinenmodell wäre gut, Quadrieren soll keinen Einfluss haben

Die Klasse P

P ist die Klasse von Sprachen, die in polynomieller Zeit durch eine *deterministische* Einband-Turingmaschine entscheidbar sind.

$$P = \bigcup_k \mathrm{TIME}(n^k)$$

- praktisch lösbare Probleme, es existieren effiziente Algorithmen
- unabhängig vom (deterministischen) Maschinenmodell
  - Turingmaschine, Registermaschine, Java-Programm etc.

Beispiel: PATH Problem

- Frage: Gibt es in einem gerichteten Graphen G einen Weg zwischen zwei Knoten s, t ?
- Es gilt zu beweisen, dass PATH $\in$ P, indem man einen Algorithmus mit polynomieller Laufzeit findet.
- Lösungsidee: Breitensuche
- Eingabe für Algorithmus: Graph G als Adjazenzmatrix oder als Liste aller Knoten und Kanten

Beispiel: PATH Problem

- Algorithmus
  - Markiere Knoten *s*.
  - Wiederhole, bis kein Knoten mehr markiert wurde: Betrachte alle Kanten von G. Wenn eine Kante *(a,b)* von einem markierten Knoten *a* zu einem noch nicht markierten Knoten *b* gefunden wurde, markiere *b*.
  - Wenn t markiert ist, akzeptiere. Sonst, weise zurück.

Beispiel: PATH Problem

- Laufzeitanalyse
  - Anweisung 1 und 3 werden nur einmal ausgeführt
  - Anweisung 2 wird maximal *m* mal ausgeführt, wobei *m* die Anzahl der Knoten bezeichnet
  - Insgesamt also 1+1+m, ein Polynom in Abhängigkeit zur Größe des Graphen (Anzahl Knoten)
  - Einzelne Schritte sind leicht in polynomieller Zeit zu implementieren.

  PATH $\in$ P

Beispiel: ggT

- Euklidischer Algorithmus (moderne Variante)
- Eingabe: x,y $\in \mathbb{N}$ in Binärdarstellung

2. Wiederhole bis y = 0
3. x = x mod y
4. Vertausche x und y
5. Ausgabe von x

Als Entscheidungsproblem formuliert: Gibt es ggT >1 ?

Beispiel: ggT

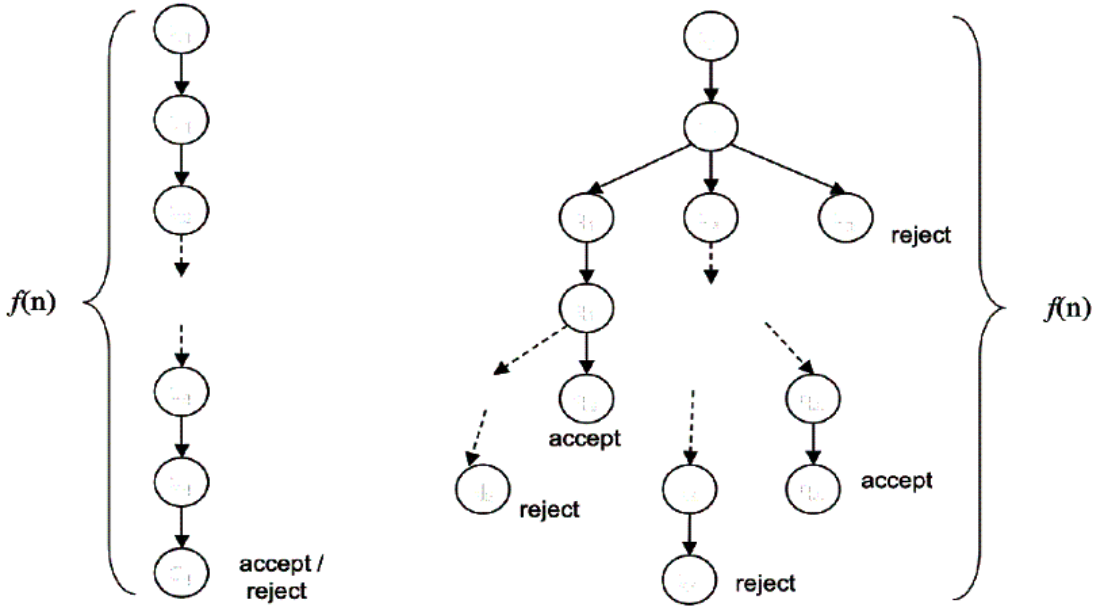
- Analyse
  - Annahme: x > y
    - nach Stufe 2: $x < y$, nach 3: $x > y$
  - Wenn $x/2 \geq y$, dann ist $x \bmod y < y \leq x/2$
  - Wenn $x/2 < y$, dann ist $x \bmod y = x - y < x/2$
    - x wird bei Ausführung von 2 mindestens halbiert
    - in 3 werden sie vertauscht, also auch y wird halbiert
  - Stufe 2 und 3 wird maximal 2log x ausgeführt, proportional zur Länge der Eingabe
  - O(n) für jede Stufe, jede Stufe braucht polynomielle Zeit also auch gesamter Algorithmus → ggT ∈ P

Nichtdeterminismus

- Deterministische Turingmaschinen sind für Probleme aus P „zuständig“, was leisten nichtdeterministische?
- Allgemeines:
  - Überführungs*relation* statt Funktion
  - jede Konfiguration kann mehrere Folgekonfigurationen haben (Baum)
  - theoretischen Maschinenmodell

Nichtdeterminismus

- Vergleich: deterministischer und nichtdeterministischer Baum von Rechenschritten



Nichtdeterministische TM

- N akzeptiert Eingabe w, wenn es mindestens einen akzeptierenden Rechenweg im Baum gibt
  - L(N) besteht aus allen w, die N akzeptiert
- Rechenzeit: Anzahl der Rechenschritte auf kürzestem Weg bei $w \in L$ und 0 bei $w \notin L$
- *worst case* Rechenzeit: $f : \mathbb{N} \rightarrow \mathbb{N}$, f(n) ist maximale Anzahl von Schritten für jeden Rechenweg bei allen Eingaben w der Länge n
  - Voraussetzung – N hält für alle Eingaben

Nichtdet. vs. Determinismus

Satz:

Sei t(n) eine Funktion mit $t(n) \geq n$. Dann gibt es für jede t(n)-zeitbeschränkte *nichtdeterministische* Einband-Turingmaschine eine äquivalente $2^{O(t(n))}$ -zeitbeschränkte *deterministische* Einband-Turingmaschine.

Nichtdet. vs. Determinismus

- Beweis: Analyse der zusätzlichen Zeit
  - Gegeben *N* und *D*
  - *N* läuft in Zeit $t(n) \geq n$.
- *D* simuliert *N*, indem sie *N*s nichtdeterministischen Baum von Rechenschritten nach akzeptierender Rechnung durchsucht

Nichtdet. vs. Determinismus

- Analyse
  - Bei Eingabe der Länge n hat jeder Ast des Baums von N maximale Länge t(n)
  - jeder Knoten (Konfiguration von N) hat maximal b Kinder
    - b ist Anzahl der möglichen Übergänge, durch Überführungsrelation vorgegeben
  - Gesamtzahl Blätter = Zahl der Äste = max. $b^{t(n)}$
  - D durchsucht Äste in O(t(n)) Zeit
  - Insgesamt: $O(t(n)\, b^{t(n)}) = 2^{O(t(n))}$

q.e.d

Die Klasse NP

- NP enthält alle von nichtdeterministischen Turingmaschinen in polynomieller Zeit lösbaren Probleme
- Basis ist unrealistischen Maschinenmodell
  - NP-Probleme gelten als nicht effizient lösbar
- Bekannte Probleme in NP sind z.B. das Rucksackproblem und das Hamiltonkreisproblem

Ausblick P vs. NP

- Offensichtlich gilt: $P \subseteq NP$
- Bis heute ungeklärte Frage:
  - $P=NP$ oder $P \neq NP$

- Mehr dazu im nächsten Vortrag