Rautavistische Algorithmen + Datenstrukturen, Berechenbarkeit, Komplexitätstheorie

(Gastvorlesung an der TU Garching, 15.7.2004)

Nils Kammenhuber
Rautavistische Universität Eschweilerhof
http://www.uni-eschweilerhof.de/cs/fb17/

Übersicht

- DEG-Graphen
  - Turingmaschinen mit DEG-Übergangsgraphen
  - Graphenalgorithmen für DEG-Graphen
- Sortieren
- Hashing
- Komplexitätsklasse NN

DEGs

Deutlich Endliche Graphen

Deutlich Endliche Graphen

- Graph $G=(V, E)$ heißt *deutlich endlich groß*, wenn gilt:

  1. $|E| \geqslant |V| \geqslant 1$

  2. G ist stark zusammenhängend.
     [Formal: $\forall v_1, v_2 \in V : \exists$ Pfad $v_1 \rightarrow \ldots \rightarrow v_2$]

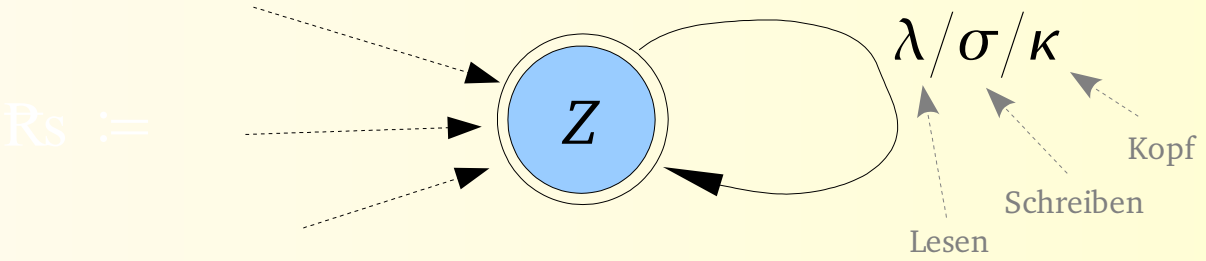
  3. „*R-Bedingung*“:
     $\exists$ Pfad $(v_1, v_2, \ldots, v_r) \wedge \exists$ Pfad $(v_r, v_{r+1}, \ldots, v_i, v_1) \Rightarrow (v_1 = v_r)$

DEG-Graphen und Halteproblem

- Satz:
  - Sei G=(V,E) DEG-Graph.
  - Sei G der Zustandsübergangsgraph von *M* (*M* ist TM).
  - Dann ist Halteproblem für *M* entscheidbar
- Beweis:
  - dreiteilig

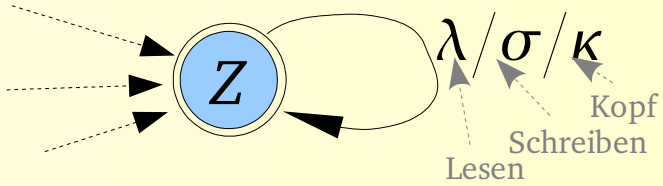
Lemma 1

- Zustandsübergangsgraph enthalte folgende Struktur ₹s:



- Wenn *Z* erreicht und Bandinhalt bekannt, dann ist Halteproblem entscheidbar (ab diesem Punkt)
- Beweis:
  - Gelesenes Zeichen sei *x*
  - Fallunterscheidung…

Beweis



- Fall 1: $\lambda \neq x \Rightarrow M$ terminiert sofort, da in Endzustand
- Fall 2: $\lambda = x$
  - Fall 2.1: κ = „L“ $\Rightarrow M$ terminiert irgendwann:
    - Wenn Zeichen nicht mehr passt oder Bandende erreicht, und weil keine andere Kopfbewegung mehr möglich
    - Band enthält endlich viele Zeichen (⇐ endliche Eingabe; endliche Anzahl Schritte ausgeführt)
  - Fall 2.2: κ = „R“ $\Rightarrow M$ terminiert irgendwann (analog 2.1)
  - Fall 2.3: κ = „N“
    - Fall 2.3.1: $\lambda = \sigma \Rightarrow M$ terminiert nicht (Endlosschleife)
    - Fall 2.3.2: $\lambda \neq \sigma \Rightarrow M$ terminiert im nächsten Schritt (Fall 1)

Lemma 2

- Jeder DEG enthält mindestens ein ₹s
- Beweis: [➳Tafel]

Lemma 3

ϵ

Algorithmus: `first-vertex`

- Universell verwendbarer Algorithmus auf DEGs

```
program first-vertex
input: DEG=(V,E) ; output: v
   assert(|V|=1)
   return(V[1])
```

- Laufzeit: O(1)
- Anwendung auf Beispiel-DEG [↣Tafelbild]

`first-vertex:`
Anwendungen auf DEGs

- Kürzester Pfad $v_1 \rightarrow v_2$
- Menge der Knoten des längsten Pfades $v_1 \rightarrow v_2$
- Größte Clique in G (normalerweise **NP**-vollständig!)
- Edge-Cover-Problem
- Grüne-Knoten-Problem:

  Wenn es einen grünen Knoten $v \in V$ gibt,
  dann finde alle grünen Knoten in V

Sortieren

Vergleichsbasiertes Sortieren
in linearer Zeit

2-way-Mergesort

- „Normaler“ Mergesort
- Rekursiv:
  - Aufteilen in 2 Listen
  - Sortieren dieser 2 Listen
  - Geordnetes Zusammenfügen („merge“) der 2 Listen
- Laufzeit: $O(n \log_2 n)$

k-way-Mergesort

- Gern für Datenbanken / große Datenmengen
- Rekursiv:
  - Aufteilen in ***k*** Listen!
  - Sortieren, Zusammenmergen wie gehabt
- Laufzeit: $O(n \log_k n)$ (!!)
  - $\log_k n$ wächst für große k sehr langsam → „fast“ $O(n)$:
  - $\log_{100}(1000)=1{,}5$; $\log_{100}(1\text{ Mio})=3$; $\log_{100}(1\text{ Mrd})=4{,}5$
- → k möglichst groß wählen

Rautavistische Informatik:
***n*-way-Mergesort**

- k möglichst groß wählen?

  → wähle $k=n$

- Laufzeit:

  $O(n \log_n n)$

  $=O(n \times 1)$

  $=O(n)$,
  also lineare Laufzeit für vergleichsbasiertes Sortieren!

Sortieren

Vergleichsbasiertes Sortieren
in konstanter Zeit:

LSORT

LSORT

```
program lsort
input: A[1...n] ; output: B[]
   if A[1]≤A[2] then:
      B[1]=A[1] ; B[2]=A[2].
   else:
      B[1]=A[2] ; B[2]=A[1].
```

- B[ ] ist eindeutig sortiert
- Vergleichsbasiertes Sortieren
- (Nachteil: in manchen Fällen verlustbehaftet, z.B. falls n > 2)

Laufzeit von LSORT (1)

- **Satz:** Die Laufzeit von LSORT ist O(1)
- **Beweis:**
  - $S := (U, g, \xi)$ Wyrre-Basis der Semantik von LSORT
    1. S ist frei von inneren Produkten (klar)
       $\Rightarrow \exists$ mindestens ein in sich geflochtenes τ in U.
       (Satz von Seeman)
    2. Jedes $\tau \times g$ ist semipositiv definit (nach Definition)
  - **Gegenannahme:**
    Laufzeit von LSORT sei $\eth\tau$, mit $\eth\tau > O(1)$
    $\Rightarrow \exists$ Differentialsperre $\eth\xi_{<g>}$ für alle Basalelemente in ${}_{g}U$
    in $\mathbb{R}^{66}$ (wg. Lemma von Kasimir-Kostonjenko)

Laufzeit von LSORT (2)

- Inkohärenzprinzip ⇒ vollständige Variablensubsitution
  - Wir können kein f höherer Ordnung finden, das hinteres Produkt $\langle \Psi, ? \rangle$ von Q assimiliert (Mülkbauer-Lemma)
  - Jegliches mittlere Produkt μ von Q bildet hyperbolische nattifftoffische Projektionsekliptik
- Ist beides der Fall, dann gibt es eine $\mathbb{R}^{66}$-spurtreue nichtdeterministische Turingmaschine M,
  die H_0 in negativer Laufzeit löst
  (Satz von Davidson und Harley)
- Umkehrschluss ⇒ kann nicht gelten ⇒ QED.

Sortieren

Ein Array besonders gut und sicher sortieren

BubbleSort

- Eigentlich ineffizienter Algorithmus; läuft in $O(n^2)$
- Hier nur aus didaktischen Gründen:

```
program bubblesort
input: A[1...n] ; output: B[]
   for i:=1 to n-1 do:
      for j:=i to n-1 do:
         if A[j]>A[j+1] then:
            swap A[j]↔A[j+1]
```

- Entfernt nach und nach alle benachbarten Inversionen
- Aber: Ist `B[ ]` wirklich (und gut) sortiert?

Problem: Ist B[] gut genug sortiert?

- Werden genügend Fehlstellungen entfernt?
- Verdächtig: `j` läuft nur ab `i`, nicht ab 1!

```
program bubblesort_hardened
input: A[1...n] ; output: B[]
for x:=1 to n-1 do:
      for i:=1 to n-1 do:
          for j:=1 to n-1 do:
              if A[j]>A[j+1] then:
                  swap A[j]↔A[j+1]
```

- ⇒ `B[]` ist jetzt besonders gut sortiert
- Vorteil: Vorhandene Rechenleistung wird ausgenutzt

Hashing

Schnelle Write-Operationen

Problem: Hashkollisionen

- Was tun, wenn $x \neq y$, aber $h(x) = h(y)$?
- Möglichkeit 1: *Listen bilden?* [↣Tafelbild]
  - u.U.: Lange Laufzeit, bis freier Platz gefunden
- Möglichkeit 2: *Multiple Probing?* [↣Tafelbild]
  - Versuche zunächst bei $h(x,0)$, dann $h(x,1)$, dann $h(x,2)$…
  - u.U.: Lange Laufzeit, bis freier Platz gefunden
- Möglichkeit 3: *Einfach überschreiben???*
  - Verlustbehaftetes Hashing
  - Ungewisser Gewinn

Lösung: Rautavistisches Hashing

- Zu hashende Objekte seien o.B.d.A. $(x_1, x_2, \ldots, x_n)$

- Rautavistische Hashfunktion h_R:

  $h_R: \{x_1, x_2, \ldots, x_n\} \to \{1, \ldots, n\}$

  $h_R(x_i) := f_R^{-1}(i)$

  wobei:

  $f_R: \mathbb{R} \times \mathbb{R} \to \mathbb{R}$

  $f_R(x, p) := x \cdot \iint (\log_x(px) - \log_x p)\, dx^2$

  [**hier:** wähle p = konstant]

- Beispiel: [➳Tafel]

Rautavistisches Hashing (Forts.)

- Vorteile:
  - Keine Hashkollisionen
  - h_R in O(1) berechenbar (sehr kleine Konstanten!)
- Nachteile:
  - Obere Schranke für n sollte bekannt sein
  - Retrieval dauert etwas länger (h_R hier nicht anwendbar!)

Hashing

Schnelles und platzsparendes Hashing

Schnell und platzsparend

- Kombiniere rautavistisches Hashing mit Möglichkeit 3
- Zeit zum Speichern: O(1)
  - konstante Faktoren sogar noch kleiner
- Platzverbrauch: O(1) bei n Elementen
- Formal: Rautavistische Hashfunktion h'_R:

  $$h'_R: \{x_1, x_2, \ldots, x_n\} \to \{1, \ldots, 1\}$$
  $$h'_R(x_i) := h_R(x_1) = 1$$

- Laufzeitbeweis: analog; hier: Strukturenzinduktion über alle ariden Gebiete ´G mit ammonitischen Radiolarien

Lalalalalala Testfolie

- Achtung! Achtung!
- Diese Folie vor dem Vortrag unbedingt rauslöschen!!!
- Urgelmurgel mörpeldörpel blurf rhmsehrmpfglbrps?

Komplexitätstheorie

*Die Komplexitätsklasse **NN***

Komplexitätshierarchie (Ausschnitt)

- …(PSPACE, EXPTIME, usw.)… ⊇
- **NP** ⊇
- …(BPP, usw.)… ⊇
- **P** ⊇
- …(NL, L usw.)… ⊇
- **NN**

Turingmaschinen

- Nichtdeterministische TM
  - Zustandsübergange nicht deterministisch
  - Kann richtige Lösung „raten“
- Definition: Laufzeit einer Turingmaschine:
  - Zeit zwischen Lesen des ersten Eingabezeichens
  - Bis zum Schreiben des letzten Ausgabezeichens

Die Klasse NN

- Gegeben: Problem P, $P \in$ **NN**
- Löse P mit Turingmaschine M:
  - M ist nichtdeterministische Maschine
  - Rate nichtdeterministisch Lösung
  - Schreibe Lösung auf Band
  - Lese Eingabe von Band
- $\Rightarrow$ Laufzeit von M ist < 0 (nach Definition)

Das Ende ist nahe!

Zusammenfassung

Zusammenfassung

Es gibt immer Leute, die keine Ironie verstehen.

Daher:

- Alle Aussagen sind:
  - Entweder richtig, aber unbrauchbar…
  - …oder falsch! (Mergesort)

- **Bitte nicht in der Klausur / Prüfung anwenden!**

ENDE

Folien (irgendwann) online unter:

http://www.uni-eschweilerhof.de/cs/fb17/