Algorithmen

Michael Kaufmann (Leitung)
Henry Förster, Axel Kuckuk (Übungsorganisation)
viele Tutoren ...

18/10/2021

Organisatorisches

Vorlesungstermine: Mo 10.30-11.45 und Di 8.15 -9.45
Besprechung der Übungsaufgaben: Übungsgruppen in Präsenz, ein paar wenige auch online.
Help Desk: Mo 12-14 Discord
Musterlösungen: Mo 16-18 Zoom

Klausur: letzte Vorlesungswoche
Nachklausur: letzte Woche vor SoSe Beginn
Teilnahme nur an Nachklausur möglich (dann keine 2. Chance!)
Übungsblätter: 50 Prozent der Punkte gefordert, Boni für Klausu

Anmeldung für Tutorien (am besten heute !!!)

Organisation: https://moodle.zdv.uni-tuebingen.de/

vorbereitende Präsenzübungen im Tutorium

Start: Ab Mittwoch dieser Woche !!

Gliederung in Themen

I. Einführung
- Motivation und Notationen
- Laufzeitanalyse, Rekursionen

II. Grundlegende Datenstrukturen
III. Graphenalgorithmen
IV. Sortieren
V. Suchen
VI. Generische algorithmische Methoden
VII. Algorithmen auf Zeichenketten

Algorithmen kommen überall vor !

- Zugverkehr:
  - Rechne einen Zugfahrplan aus
  - Weise jeder Zuglinie eine physische Lok + Waggons zu
  - Berücksichtige Zugausfälle/Verspätungen, um neue Empfehlungen zu geben
- Börse
  - Löse schwierige kombinatorische Probleme
  - Zeitkritisch ! Gute Lösungen in Millisekunden !
- Biologie
  - Sequenziere ein Genom !
  - Analysiere und Vergleiche mit anderen !

Algorithmen kommen überall vor !

- Tutorienverwaltung:
  - 400 Studis, werden verteilt auf 15 Gruppen
  - Jede hat zeitliche Präferenzen
  - Berechne Zuweisung, so dass jede zufrieden ist.
- Internet:
  - Netzwerkprotokolle, möglichst robust gegen Fehler
  - Routing Standards $\rightarrow$ Vorteile/Nachteile verschiedener kürzester Wege !
- Firmen wie Google:
  - Indiziere alle Webseiten !
  - Wie kann man Speicherkapazität sparen, ohne Redundanz aufzugeben ?
  - Beantworte Suchanfragen hoch qualitativ und schnell

Informatik ist viel mehr als Programmieren ...

- Viele Leute können programmieren (oder denken es). Braucht kein Informatiker zu sein, um Java Code zu schreiben.
- Um praktische Probleme zu lösen, braucht es mehr als die Syntax einer PS. Man kann nicht einfach hinsitzen und Lösungen implementieren.
- Was ein Informatiker können sollte:
  - Gegeben ein praktisches Problem, 'abstrahiere' von allen schmutzigen, anwendungsabhängigen Details.
  - Werkzeugkasten !
  - Analyse: Wie schwierig ist das Problem? Wie gut ist mein Ansatz? Geht es auch besser, und mit welchem Aufwand ?
  - Beschreibe Lösung genau genug, so dass ein Programmierer sie umsetzen kann.

Warum ist das wichtig für Sie?

- Viele Informatiker laufen unweigerlich in algorithmische Probleme im späteren Beruf
- Sie sollten es wenigstens merken, wenn es passiert.
- Sie sollten wenigstens ein Grundverständnis haben, wie solche Probleme angegangen werden sollten
- Dieses Grundverständnis ist notwendig, um vernünftige Lösungen zu finden (sogar wenn Sie in der Literatur nachlesen)

Literatur

- S. Dasgupta, Ch. Papadimitriou, U. Vazirani. Algorithms, McGraw-Hill, 2008.
- T.Cormen, Ch.Leiserson, R.Rivest, C.Stein: Introduction to algorithms and data structures. 3rd edition. MIT Press, 2009.
- J. Kleinberg, E. Tardos: Algorithm Design. 2005.
- K. Mehlhorn, P. Sanders, M. Dietzfelbinger: Algorithmen und Datenstrukturen. Springer, 2014.
- Außerdem: Vöcking et al.: Taschenbuch der Algorithmen. Springer, 2008. (Engl. Algorithms unplugged)
- Sowie: J. MacCormick: 9 Algorithms that changed the future, Princeton University Press, 2012.

Was sind Algorithmen und wie geht Algorithmenanalyse ?
Wikipedia: Ein A. ist eine eindeutige Handlungsvorschrift zur Lösung eines Problems oder einer Klasse von Problemen.
A's bestehen aus endlich vielen, wohldefinierten Einzelschritten.
Damit können sie zur Ausführung in ein Computerprogramm implementiert, aber auch in menschlicher Sprache formuliert werden. Bei der Problemlösung wird eine bestimmte Eingabe in eine bestimmte Ausgabe überführt.

Name: – Buch von Al Khwarizmi (9. Jhd.) über Dezimalsystem.
– Genaue eindeutige Beschreibungen numerischer Operationen wie Addition, Dividieren, lineare Gleichungen
– latein. Übersetzung (12. Jhd): Algoritmi de numero Indorum

Erstes Beispiel aus Dasgupta: Fibonacci Zahlen

$$F(n) = \left\{ \begin{array}{ll} 0, & \text{falls } n = 0 \\ 1, & \text{falls } n = 1 \\ F(n-1) + F(n-2), & \text{falls } n \geq 2 \end{array} \right\}$$

Ergibt die Folge 0,1,1,2,3,5,8,13,21, ...

Zahlen wachsen sehr schnell. Für $n \geq 6$ haben wir
$2^{n/2} \leq F(n) \leq 2^n$.

Wie sehen potentielle Algorithmen aus ?
$\rightarrow$ Schauen wir uns mal 2 an. Es sollte einfach sein, oder ? ...

rekursiv oder nicht-rekursiv ! Probieren Sie beides !

Algorithmus 1: nicht-rekursive einfache Schleife

Input: $n \in \mathbb{N}$
Output: Die n-te Fibonacci-Zahl F_n

1 **Algorithm:** FibLoop(n)
2 Allokiere F als ein Array der Länge $n + 1$;
3 $F[0] \leftarrow 0$;
4 $F[1] \leftarrow 1$;
5 **for** $i \in \{2, \ldots, n\}$ **do**
6 $\quad F[i] \leftarrow F[i-1] + F[i-2]$;
7 **end**
8 **return** $F[n]$

Algorithmus 2: Rekursion

Input: $n \in \mathbb{N}$
Output: Die n-te Fibonacci-Zahl F_n

```
1 Algorithm: FibRekursive(n)
2 if n = 0 then
3 |   return 0
4 end
5 if n = 1 then
6 |   return 1
7 end
8 return FibRekursive(n − 1) + FibRekursive(n − 2)
```

WELCHER ALGORITHMUS IST BESSER ?
WAS IST DAS RICHTIGE ARGUMENT ?

Laufzeiten der Fibonacci Algorithmen:
Zählen die 'Elementaroperationen'

FibLoop(n)

- Alloziere ein Array der Länge n, entweder 1 OP (Pointer auf den Anfang des Arrays, oder $<= n$ OPs (Initialisiere mit Nullen)
- 2 Elementaroperationen für 2 Zuweisungen.
- In jedem Schleifendurchlauf addieren wir 2 Zahlen.
  Da $2^{n/2} \leq F(n) \leq 2^n$ für $n \geq 6$, ist $F(n)$ etwa n Bits lang.
- Schleife wird $(n-1)$- mal durchlaufen.

Anzahl der Elementaroperationen: $1 + 2 + (n-1) \cdot n \approx n^2$

Laufzeiten der Fibonacci Algorithmen:
Zählen die 'Elementaroperationen'

FibRekursive(n)

- Anfangs jeweils 1 Elementaroperation
- Dann eine Addition und die Zeit für die rekursiven Aufrufe.
- Also $T(n) = 4 + T(addition) + T(n-1) + T(n-2) \geq 4 + T(n-1) + T(n-2)$
- Vergleich mit Definition von F_n zeigt, dass $T(n) \geq F_n$.
- Wegen $F_n \geq 2^{n/2}$ für $n \geq 6$, wir erhalten $T(n) \geq 2^{n/2}$.

Laufzeiten der Fibonacci Algorithmen: Vergleich

- Laufzeit von FibRekursive ist exponentiell in n. Also ist die Laufzeit sogar für mittelgroße n zu groß für alle denkbare Computer.
- Laufzeit von FibLoop ist quadratisch in n. Okay für relativ große n, aber nicht toll. Immerhin 'polynomiell'.
- Geht es besser als mit FibLoop ? Übungsblatt 1.

Was lernen wir daraus ?

- Sogar für ganz einfache Probleme, bisschen unterschiedliche Ansätze können zu riesigen Unterschieden führen.
- Schnelle Implementierungen sind nicht immer offensichtlich.

Was lernen wir daraus ?

- Sogar für ganz einfache Probleme, bisschen unterschiedliche Ansätze können zu riesigen Unterschieden führen.
- Schnelle Implementierungen sind nicht immer offensichtlich.

Zweck der Vorlesung:

- Sie lernen schnelle elegante Algorithmen für einige Grundprobleme kennen
- Sie erhalten Werkzeuge, um Algorithmen zu analysieren und rauszufinden, ob sie 'gut' sind
- Sie lernen 'algorithmisches Denken', was Sie zum Entwickeln neuer Algorithmen brauchen