Wo sind wir hier?

Vorlesungstitel:

Theoretische Informatik III

In dieser Einheit 0 klären wir Organisatorisches, wie:

Namen der beteiligten Dozenten; Termine, Übungen, Scheinbedingungen etc.

Die Vorlesung richtet sich in erster Linie an die
Bachelor-Studiengänge des Fachbereichs Informatik:

B.Sc. Informatik, B.Sc. Softwaretechnik, B.Sc. Data Science

und hier an Studierende des dritten Semesters.

Dozenten

Vorlesung:

Dr. Manfred Kufleitner

Privatdozent am Institut für Formale Methoden der Informatik (FMI)
Universitätsstr. 38, Raum 1.160, Tel. +49 711 685 88 231
kufleitner@fmi.uni-stuttgart.de, www.fmi.uni-stuttgart.de/ti/team/kufleitner

Folien von Dr. Ulrich Hertrampf

Übungsbetrieb:

Jan Philipp Wächter

Wissenschaftlicher Mitarbeiter am FMI
Universitätsstr. 38, Raum 1.112, Tel. +49 711 685 88 410
jan-philipp.waechter@fmi.uni-stuttgart.de, www.fmi.uni-stuttgart.de/ti/team/waechter

Planung der Vorlesung (1)

Die Einheiten 1-7 beinhalten den ersten Abschnitt der Vorlesung:

Einheit 1 (heute noch) zum Begriff des Algorithmus und zu Rekursion

Einheit 2 (11.11.) zum Entwurfsprinzip Divide and Conquer

Einheit 3 (11.11.) zum Entwurfsprinzip Dynamisches Programmieren

Einheit 4 (12.11.) zum Entwurfsprinzip Backtracking / Branch and Bound

Einheit 5 (12.11.) zum Entwurfsprinzip Greedy (gierige Algorithmen)

Einheit 6 (später) zum Begriff der Randomisierten Algorithmen

Einheit 7 (19.11.) zum sehr nützlichen Mastertheorem

Planung der Vorlesung (2)

Die Einheiten 8-17 bilden den Abschnitt 2.

Im zweiten Abschnitt behandeln wir eine Reihe wichtiger Algorithmen, die mit den vorher eingeführten Entwurfsprinzipien entworfen werden können. Unter anderem sind das Algorithmen

zur Matrixmultiplikation

zum Sortieren

für das TSP und für das String-Matching Problem

und noch viele andere.

Fünf Vorlesungen – Termine (voraussichtlich): 25.11. - 26.11. - 3.12. - 9.12. - 10.12.

Planung der Vorlesung (3)

Im dritten Abschnitt (Einheiten 18-21) behandeln wir den Bereich der algebraischen und zahlentheoretischen Algorithmen.

Damit können wir gleichzeitig den ersten Teil (Algorithmik) abschließen und den zweiten Vorlesungsteil (Diskrete Strukturen) eröffnen...

Zentrale Themen sind hier:

Der Euklidsche Algorithmus

und

Der Chinesische Restsatz

Termine: 17.12. und 7.01.

Planung der Vorlesung (4)

Die Einheiten 22-40 sind Kapitel 4. Dieser Teil der Vorlesung entspricht zu großen Teilen dem zweiten Teil der früheren Vorlesung „Logik und Diskrete Strukturen“.

Eine detaillierte Vorschau dieses Teils werden wir voraussichtlich zu Beginn der Einheit 22 angeben.

Ganz am Ende werden wir in Einheit 41 noch einmal einen Überblick über alle Inhalte des Semesters präsentieren.

Termine: 13.01. bis Semesterende (plus Einschub Randomisierte Algorithmen)

Was erwarten wir von Ihnen?

Arbeitstechnisch:

1. Vorlesung vorbereiten (immer!)
2. In der Vorlesung aufpassen, mitdenken – auch fragen
3. Nacharbeiten, offene Fragen klären, Verständnis überprüfen
4. Übungen grundsätzlich selbst bearbeiten !!!

Organisatorisch:

- Prüfungsvorleistung: Schein (Scheinkriterien ?)
- Ablegen der Prüfung möglichst direkt nach dem Semester
- Informieren Sie sich über die für Sie gültige Prüfungsordnung!

Der Algorithmenbegriff

Was ist eigentlich ein Algorithmus?

Diese Frage haben wir schon in der Theo.Inf. II Vorlesung zu beantworten versucht. Dabei haben wir gelernt, dass die These von Church allgemein anerkannt ist, nach der man davon ausgehen kann, dass wir wahlweise mit Turingmaschinen oder mit beliebigen (allgemeinen) Programmiersprachen arbeiten können.

Allen sogenannten „Konkretisierungen des Algorithmenbegriffs“ gemeinsam ist:

Ein Algorithmus ist eine schrittweise auszuführende Vorschrift, bei der jeder auszuführende Schritt tatsächlich in endlicher Zeit und auf eindeutig definierte Weise ausgeführt werden kann.

Zum Algorithmenbegriff

Wir gehen grundsätzlich davon aus, dass vor Beginn eines *Ablaufs* des gegebenen Algorithmus eine *Eingabe* bereitgestellt wird, auf die die ausführende *Maschine* während ihrer Arbeit zugreifen kann.

Die Eingabe wird häufig in Form eines Wortes über einem Alphabet angegeben. Es kann sich aber ebensogut um eine Zahl, einen Graph, eine Matrix, eine Kombination hiervon oder andere ähnliche Dinge handeln.

Grundsätzlich sollte eine Eingabe eine *Länge* $n \in \mathbb{N}$ haben.

Mit $|x|$ bezeichnen wir die Länge der Eingabe x.

Worst-case Analyse

Die worst-case Analyse haben wir bereits in der Vorlesung Theoretische Informatik II kennengelernt.

Wir rekapitulieren nochmal:

$$time_M(x) = \text{ Anzahl der Schritte von } M \text{ bei Input } x$$

d.h. $time_M : \mathbb{N} \to \mathbb{N} \cup \{\infty\}$

$$time_M(n) = \max_{|x| \leq n} \{time_M(x)\}$$

Analog werden Funktionen für nichtdeterministische und/oder platzbeschränkte Maschinen definiert:

$$ntime_M \qquad dspace_M \qquad nspace_M$$

Wir werden meist über *Algorithmen* sprechen, daher entsprechend $time_A, \ldots, nspace_A$.

Average-case Analyse

Um über *Durchschnittskomplexität* – oder *Komplexität im Mittel* – also über eine sogenannte Average-case Rechenzeit sprechen zu können, gehen wir üblicherweise davon aus, dass alle Eingaben der Länge n mit gleicher Wahrscheinlichkeit auftreten.

Sie sind also im mathematischen Sinn gleichverteilt.

Dann ist $\mathit{av\text{-}time}_A(n) = E[T_A(n)]$, wobei $T_A(n)$ bei gegebenem Algorithmus A die Zufallsvariable ist, die beim Zufallsexperiment

„Führe A auf zufälliger Eingabe der Länge n aus“

durch die Laufzeit gegeben ist. Das heißt:

$$\mathit{av\text{-}time}_A(n) = \frac{1}{N} \cdot \sum_{|x|=n} \mathit{time}_A(x)$$

mit $N = |\{x : \ |x| = n\}|$.

Beispiel: Maximumberechnung

Wir betrachten den folgenden Algorithmus, der das Maximum von n Eingaben (in einem Array $a[\,]$) ermittelt:

```
max := a[1];                          Aufwand c1
FOR i := 2 TO n DO
    IF a[i] > max THEN                Aufwand c2
        max := a[i];                  Aufwand c3
```

Wir haben vier Zeilen mit Aufwand c_1 für Zeile 1, Aufwand c_2 für Zeilen 2 und 3, sowie Aufwand c_3 für die letzte Zeile.

Im worst-case ergibt sich als Rechenzeit $c_1 + (n-1)(c_2 + c_3)$.

Average case? Die W'keit, Zeile 3 ausführen zu müssen, ist abhängig von i.
Man kann sich leicht überzeugen, dass sie genau $1/i$ ist. (Bitte überprüfen!)
Dadurch ergibt sich über die harmonische Reihe als erwartete Rechenzeit
für große n ungefähr $c_1 + (n-1)c_2 + ((\ln n) - 0.42)c_3$.

Landau-Symbole

Die fünf sogenannten *Landau-Symbole* sollten Ihnen bereits bekannt sein:

$O(f)$ ist eine Klasse von Funktionen von $\mathbb{N}$ nach $\mathbb{N}$, zu der die Funktion g gehört, wenn Konstanten N_0 und $c > 0$ existieren mit:

$$g(n) \leq c \cdot f(n) \text{ für alle } n \geq N_0$$

Wird in der Ungleichung das $\leq$-Zeichen durch $\geq$ ersetzt, d.h.

$$g(n) \geq c \cdot f(n) \text{ für alle } n \geq N_0$$

dann ist g in der Klasse $\Omega(f)$.

Die Funktionenklasse $\Theta(f)$ ist $O(f) \cap \Omega(f)$.

$o(f)$ enthält die Funktion g, falls für alle $c > 0$ ein N_0 existiert, sodass für alle $n \geq N_0$ die Ungleichung $g(n) \leq c \cdot f(n)$ gilt.

Für $\omega(f)$ muss analog $f(n) \leq c \cdot g(n)$ erfüllt sein.

Rekursion und Rekursionsgleichungen

Rekursion spielt in der Algorithmentheorie eine große Rolle!

Gegeben: Ein Problem, das von einem Parameter abhängt.
Löse dieses Problem für Parameterwert n unter
Zuhilfenahme der Lösung für Parameterwert m
(üblicherweise $m < n$).

(Manchmal auch $m_1, \ldots, m_k$, wobei alle $m_i \neq n$)

Beispiele:

`sort`$(a_1, \ldots, a_n)$: `IF` $n > 1$ `THEN` `sort`$(a_1, \ldots, a_{n-1})$;
`füge` a_n `ein.`

`nsort`$(a_1, \ldots, a_n)$: `FIND` j `WITH` $a_j \geq a_i$ `für alle` i;
`SWAP`(a_j, a_n); `nsort`$(a_1, \ldots, a_{n-1})$.

Realistischere Beispiele:

Fibonacci-Zahlen, Euklidscher Algorithmus,...