Zusammenfassung Algo-1 [im SS2021]

Version 0.0.1

Julian Keck
uwgmn

20. April 2021

Inhaltsverzeichnis

1 Kapitel 1 - Algorithmen

1.1 Begriffsdefinition

- Ein Algorithmus ist eine Rechenvorschrift zur Lösung eines Problems, bestehend aus endlich vielen, wohldefinierten Einzelschritten

1.2 Eigenschaften eines Algorithmus

- bestitzt Ein und Ausgabe
- Verfahren muss eindeutig beschreibbar sein (**Finitheit**)
- Jeder Schritt muss ausführbar sein (**Ausführbarkeit**)
- Verfahren darf nur endlich viel Speicher belegen
- Verfahren darf nur endlich viele Schritte benötigen

Algorithmen bearbeiten (strukturierte) Daten, daher sind Algorithmen und Datenstrukturen eng verknüpft.

1.3 Vergleich von Algorithmen

1.3.1 Verbrauchte Rechenzeit

- Laufzeit des Programms selbst
- In der Praxis zusätzlich: Zeit für Ein-/Ausgabe, etc...

1.3.2 Verbrauchter Speicher für Programm und Datenstrukturen

- Platz für das Programm selbst
- Platz für statische Datenstrukturen
- Platz für dynamsiche Datenstrukturen

1.3.3 Analyse von Algorithmen

- formale Betrachtung des Algorithmus
- theoretische Abschätzung der Laufzeit oder des Speicherbedarfs abhängig von der Eingabegröße

1.4 Pseudocode

Zur Beschreibung von Algorithmen wird oftmals Pseudocode verwendet, da dieser klar von Programmcode unterschieden werden kann und dieser Redundanzen oder Unklarheiten, die durch "Besonderheiten"einer bestimmten Programmiersprache entstehen, vorbeugt.

1.5 Algorithm Engineering

Klassiche Algorithmik beschränkt sich auf die Theorie
Beim tatsächlichen Design von Algorithmen sollte neben der theoretischen Analyse aber auch immer eine experimentelle Evaluierung stattfinden. Das Gebiet «Algorithm Engineering» schließt also die Lücke zwischen Theorie und Praxis.

1.6 Korrektheit / Design by Contract

1.6.1 Contracts

Auch «Verträge» genannt, bestehen aus:

- Vorbedingungen (preconditions): Zusicherungen, die beim Aufruf (der Funktion/Algorithmus) einzuhalten sind.
- Nachbedingungen (postconditions): Zusicherungen, die am Ende (der Funktion/Algoirthmus) gelten

1.6.2 Assertions

Aussagen über den Zustand der Programmausführung

- Vorbedingung: Bedingung für korrektes Funktionieren einer Funktion.
- Nachbedingung: Leistungsgarantie einer Funktion, falls Vorbedingung erfüllt.

1.6.3 Invarianten

Zusicherungen, die an einer bestimmten Stelle (im Programm/Algoithmus) gelten

- Schleifeninvarianten: Gelten vor/nach jeder Ausführung des Schleifenkörpers.
- Datenstrukturinvarianten: Gelten vor/nach jedem Aufruf einer Operation auf einem abstrakten Datentypen.

Invarianten sind zentrales Mittel für Korrektheitsbeweise im Algorithmen-/Programmentwurf.

2 Laufzeitanalyse von Algorithmen

2.1 Asymptotischer Aufwand / O-Kalkül

Bereits aus GBI bekannt:
$O(f(n)) = \{g(n) \mid \exists c > 0 : \exists n_0 \in \mathbb{N}_+ : \forall n \geq n_0 : g(n) \leq c \cdot f(n)\}$
$\Theta(f(n)) = \{g(n) \mid \exists c > 0 : \exists n_0 \in \mathbb{N}_+ : \forall n \geq n_0 : g(n) = c \cdot f(n)\}$
$\Omega(f(n)) = \{g(n) \mid \exists c > 0 : \exists n_0 \in \mathbb{N}_+ : \forall n \geq n_0 : g(n) \geq c \cdot f(n)\}$
Zusätzlich definieren wir
$o(f(n)) = \{g(n) \mid \forall c > 0 : \exists n_0 \in \mathbb{N}_+ : \forall n \geq n_0 : g(n) \leq c \cdot f(n)\}$, also «echt kleiner/langsamer»,
sowie
$\omega(f(n)) = \{g(n) \mid \forall c > 0 : \exists n_0 \in \mathbb{N}_+ : \forall n \geq n_0 : g(n) \geq c \cdot f(n)\}$, also «echt größer/schneller».

2.1.1 Rechenregeln im O-Kalkül

- $\forall c > 0 : \ cf(n) \in \Theta(f(n))$
- $\sum_{i_0}^{k} a_i n^i \in O(n^k)$ (Der größte Exponent im Polynom entscheidet)
- $f(n) + g(n) \in \Omega(f(n))$
- $f(n) + g(n) \in O(f(n))$, falls $g(n) \in O(f(n))$
- $O(f(n)) \cdot O(g(n)) = O(f(n) \cdot g(n))$

2.1.2 Rechenregeln für Algorithmen

- $T(I; I') = T(I) + T(I')$
- $T(\texttt{if } C \texttt{ then } I \texttt{ else } I') \in O(T(C) + max(T(I), T(I'))$
- $T(I; I') = T(I) + T(I')$

2.2 Master Theorem - Laufzeit *rekursiver* Algorithmen

Beschreibe $r(n) \begin{cases} \texttt{a} & \text{falls } n = 1 \text{ (Basisfall)} \\ \texttt{c} \cdot n + \texttt{d} \cdot r(\frac{n}{\texttt{b}}) & \text{falls } n > 1 \text{ (teile und herrsche)} \end{cases}$ die Laufzeit eines Teile und Herrsche-Algorithmus und außerdem gelte $\texttt{a}, \texttt{b}, \texttt{c}, \texttt{d} > 0$ und $n = \texttt{b}^k$ für $k \in \mathbb{N}_+$.
Dann gilt:

$$r(n) \in \begin{cases} \Theta(n) & \text{falls d<b} \\ \Theta(n \cdot log(n)) & \text{falls d=b} \\ \Theta(n^{log_b(d)}) & \text{falls d>b} \end{cases}$$

Die Werte a und c sind also für die asymptotische Laufzeit irrelevant.

2.2.1 Beispiele

- $d < b$: Median bestimmen
- $d = b$: Sortieralgorithmen wie mergesort, quicksort
- $d > b$: rekursive Multiplikation, Karatsuba-Ofman-Multiplikation