Karlsruhe Institute of Technology

Tutorium 1:
Organisatorisches, O-Kalkül, Schleifeninvarianten

Matthias Schimek | 3. Juni 2017

TUTORIUM ZUR VORLESUNG ALGORITHMEN I IM SS17

Gliederung

Organisatorisches - Tutorium

- Matthias Schimek, 1.Semester Info Master (vermutlich ;))
- **E-Mail:** `matthias_schimek@gmx.de`
- Schreibt mich für Anregungen, Fragen und Feedback einfach an
- **Folien:** `https://github.com/mschimek/Algo_Tutorium`

Organisatorisches - Vorlesung

- **Website:** `https://crypto.iti.kit.edu/index.php?id=799`
- **ILIAS-Forum:** `https://ilias.studium.kit.edu/goto.php?target=crs_671935&client_id=produktiv`
- **Mailingliste:** `https://lists.ira.uni-karlsruhe.de/mailman/listinfo/algorithmeni`
- **Feedback-Kasten** auf VL-Website

Organisatorisches - Übungsblätter

- **Abgabe:** spätestens bis Dienstag 12:45 Uhr eine Woche nach Ausgabe
- Abgabe ist in Zweiergruppe möglich
  - Beide müssen im selben Tut sein
  - Eine Abgabe pro Gruppe
  - Hat jemand noch keinen Partner? ...
- Für die Abgabe das offizielle Deckblatt verwenden

Übungsblätter - Organisatorisches

- Voraussichtlich 11 Blätter à 18 Punkte
- Es gibt einen Klausurbonus

Klausurbonus

- $>$25% der Punkte $\Rightarrow$ 1 Bonuspunkt
- $>$50% der Punkte $\Rightarrow$ 2 Bonuspunkt
- $>$75% der Punkte $\Rightarrow$ 3 Bonuspunkt

Übungsblätter - Stil

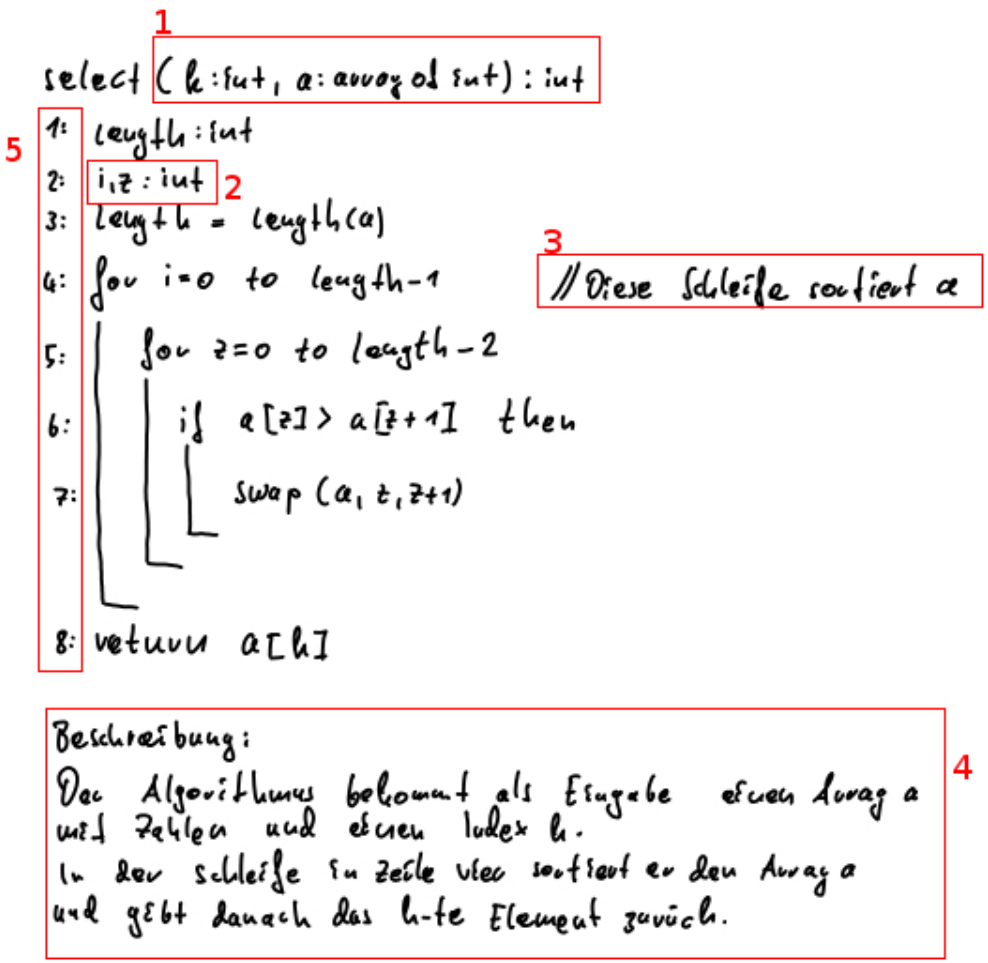
- Abgabe handschriftlich und halbwegs ordentlich ;)
- keinen Blei- oder Rotstift verwenden
- Aufgaben möglichst kurz und prägnant beantworten
- **Beweise und Pseudocode:** wichtige Idee aufschreiben

Übungsblätter - Pseudocode

Wichtige Punkte für das Schreiben von Pseudocode:

- Eingabe- und Rückgabeparameter des Algorithmus benennen
- Komplexe Schritte mit Kommentaren versehen
- Mathematische Notation verwenden, um Pseudocode kompakt zu halten
- Kurze Beschreibung der wesentlichen Idee des Algorithmus' in Textform

Ein Beispiel



1: Genaue Beschreibung der Eingabe und Ausgabe und deren Typen.

2: Sinnvolle Wahl von Indizes (i und j kann man meistens schlecht unterscheiden).

3: Einzelne Zeilen sind sinnvoll auskommentiert.

4: Eine kleine Beschreibung, was der Algorithmus tut, hilft beim Verständnis.

5: Wenn in der Beschreibung auf einzelne Zeilen verwiesen wird, dann sollte man die Zeilen auch durchnummerieren.

Asymptotik und O-Kalkül

Asymptotik und O-Kalkül

- Wollen möglichst allgemeine, aber einfache Aussage über Laufzeit
- **Asymptotik:**
  - es existiert n_0, sodass für all $n > n_0$ gilt $A(n)$ ...
  - bekommen Aussage für *fast* alle Eingabegrößen
- **Beispiel:**
  - Algorithmus A mit Laufzeit $A(n) = \log(n)$
  - Algorithmus B mit Laufzeit $B(n) = n^{0,0001}$
  - Für $n = 10^{100}$: $A(n) = 100$ $B(n) = 10^{0,01} \approx 1.02329299228$
  - Für $n = 10^{10^{100}}$: $A(n) = 10^{100}$ $B(n) = 10^{10^{96}} = 10^{1000000000\ldots00000}$
  - $\Rightarrow$ Für *fast* alle n ist Algo A schneller als Algo B
- Asymptotische Analyse $\sim$ DENKE IM GROSSEN

Asymptotik und O-Kalkül

- Wollen möglichst allgemeine, aber einfache Aussage über Laufzeit
- **Asymptotik:**
  - es existiert n_0, sodass für all $n > n_0$ gilt $A(n)$ ...
  - bekommen Aussage für *fast* alle Eingabegrößen
- **Beispiel:**
  - Algorithmus A mit Laufzeit $A(n) = \log(n)$
  - Algorithmus B mit Laufzeit $B(n) = n^{0,0001}$
  - Für $n = 10^{100}$: $A(n) = 100$ $B(n) = 10^{0,01} \approx 1.02329299228$
  - Für $n = 10^{10^{100}}$: $A(n) = 10^{100}$ $B(n) = 10^{10^{96}} = 10^{1000000000\ldots00000}$
  - $\Rightarrow$ Für *fast* alle n ist Algo A schneller als Algo B
- Asymptotische Analyse $\sim$ DENKE IM GROSSEN

Asymptotik und O-Kalkül

- Wollen möglichst allgemeine, aber einfache Aussage über Laufzeit
- **Asymptotik:**
  - es existiert n_0, sodass für all $n > n_0$ gilt $A(n)$ ...
  - bekommen Aussage für *fast* alle Eingabegrößen
- **Beispiel:**
  - Algorithmus A mit Laufzeit $A(n) = \log(n)$
  - Algorithmus B mit Laufzeit $B(n) = n^{0,0001}$
  - Für $n = 10^{100}$: $A(n) = 100$ $B(n) = 10^{0,01} \approx 1.02329299228$
  - Für $n = 10^{10^{100}}$: $A(n) = 10^{100}$ $B(n) = 10^{10^{96}} = 10^{1000000000\ldots00000}$
  - $\Rightarrow$ Für *fast* alle n ist Algo A schneller als Algo B
- Asymptotische Analyse $\sim$ DENKE IM GROSSEN

Asymptotik und O-Kalkül

- Wollen möglichst allgemeine, aber einfache Aussage über Laufzeit
- **Asymptotik:**
  - es existiert n_0, sodass für all $n > n_0$ gilt $A(n)$ ...
  - bekommen Aussage für *fast* alle Eingabegrößen
- **Beispiel:**
  - Algorithmus A mit Laufzeit $A(n) = \log(n)$
  - Algorithmus B mit Laufzeit $B(n) = n^{0,0001}$
  - Für $n = 10^{100}$: $A(n) = 100$ $B(n) = 10^{0,01} \approx 1.02329299228$
  - Für $n = 10^{10^{100}}$: $A(n) = 10^{100}$ $B(n) = 10^{10^{96}} = 10^{1000000000\ldots00000}$
  - $\Rightarrow$ Für *fast* alle n ist Algo A schneller als Algo B
- Asymptotische Analyse $\sim$ DENKE IM GROSSEN

Asymptotik und O-Kalkül

- Wollen möglichst allgemeine, aber einfache Aussage über Laufzeit
- **Asymptotik:**
  - es existiert n_0, sodass für all $n > n_0$ gilt $A(n)$ ...
  - bekommen Aussage für *fast* alle Eingabegrößen
- **Beispiel:**
  - Algorithmus A mit Laufzeit $A(n) = \log(n)$
  - Algorithmus B mit Laufzeit $B(n) = n^{0,0001}$
  - Für $n = 10^{100}$: $A(n) = 100$ $B(n) = 10^{0,01} \approx 1.02329299228$
  - Für $n = 10^{10^{100}}$: $A(n) = 10^{100}$ $B(n) = 10^{10^{96}} = 10^{1000000000\ldots00000}$
  - $\Rightarrow$ Für *fast* alle n ist Algo A schneller als Algo B
- Asymptotische Analyse $\sim$ DENKE IM GROSSEN

O-Kalkül

- Definiert Komplexitätsklassen (= Menge von Funktionen)
- Wozu? - Hilfreich bei der Laufzeitanalyse von Algorithmen
  - Abschätzung der asymptotischen Laufzeit (Eingabelänge $\to \infty$)
  - Abstraktion: konstante Faktoren nicht wichtig
  - Abstraktion: asympt. langsamer wachsende Terme werden ignoriert

$$3 \cdot n^3 + 4,0225 \cdot n^2 + 1,5 \cdot n \cdot \log(n) + \log(\log(n)) \in \mathcal{O}(n^3)$$

  - $\Rightarrow$ Macht Formel für die Laufzeit handlich

- Vereinbarung: betrachtete Funktionen $f : \mathbb{N} \to \mathbb{N}$

- $\mathcal{O}(f(n)) =$ Menge aller Funkt. (asympt.) höchstens so groß wie f
- $\Omega(f(n)) =$ Menge aller Funkt. (asympt.) mindestens so groß wie f
- $\Theta(f(n)) =$ Schnitt von $\mathcal{O}(f(n))$ und $\Omega(f(n))$

- $o(f(n)) =$ Menge aller Funkt. (asympt.) echt kleiner als f
- $\omega(f(n)) =$ Menge aller Funtk. (asympt.) echt größer als f

- Vereinbarung: betrachtete Funktionen $f : \mathbb{N} \to \mathbb{N}$

- $\mathcal{O}(f(n)) =$ Menge aller Funkt. (asympt.) höchstens so groß wie f
- $\Omega(f(n)) =$ Menge aller Funkt. (asympt.) mindestens so groß wie f
- $\Theta(f(n)) =$ Schnitt von $\mathcal{O}(f(n))$ und $\Omega(f(n))$

- $o(f(n)) =$ Menge aller Funkt. (asympt.) echt kleiner als f
- $\omega(f(n)) =$ Menge aller Funtk. (asympt.) echt größer als f

O-Kalkül - Aufgaben

Beweise oder widerlege:

- $n^2 \in o(n^3)$
- $n^3 \in \Omega(n^2), \quad n \in \Theta(\sqrt{n}), \quad 2^{n+1} \in \mathcal{O}(2^n)$
- $\forall c_1, c_2 \in \mathbb{N}_+ \forall f : \mathbb{N} \to \mathbb{N}_+ : c_1 \cdot f(n) + c_2 \in \mathcal{O}(f(n))$
- $\forall c_1, c_2 \in \mathbb{N}_+ \forall f : \mathbb{N} \to \mathbb{N} \ : c_1 \cdot f(n) + c_2 \in \mathcal{O}(f(n))$
- $\forall c \in \mathbb{N} \forall f : \mathbb{N} \to \mathbb{N} : (f(n))^c \in \mathcal{O}(f(n))$

O-Kalkül - Aufgaben

Beweise oder widerlege:

- $n^2 \in o(n^3)$
- $n^3 \in \Omega(n^2), \quad n \in \Theta(\sqrt{n}), \quad 2^{n+1} \in \mathcal{O}(2^n)$
- $\forall c_1, c_2 \in \mathbb{N}_+ \forall f : \mathbb{N} \to \mathbb{N}_+ : c_1 \cdot f(n) + c_2 \in \mathcal{O}(f(n))$
- $\forall c_1, c_2 \in \mathbb{N}_+ \forall f : \mathbb{N} \to \mathbb{N} \ : c_1 \cdot f(n) + c_2 \in \mathcal{O}(f(n))$
- $\forall c \in \mathbb{N} \forall f : \mathbb{N} \to \mathbb{N} : (f(n))^c \in \mathcal{O}(f(n))$

O-Kalkül - Aufgaben

Beweise oder widerlege:

- $n^2 \in o(n^3)$
- $n^3 \in \Omega(n^2), \quad n \in \Theta(\sqrt{n}), \quad 2^{n+1} \in \mathcal{O}(2^n)$
- $\forall c_1, c_2 \in \mathbb{N}_+ \forall f : \mathbb{N} \to \mathbb{N}_+ : c_1 \cdot f(n) + c_2 \in \mathcal{O}(f(n))$
- $\forall c_1, c_2 \in \mathbb{N}_+ \forall f : \mathbb{N} \to \mathbb{N} \ : c_1 \cdot f(n) + c_2 \in \mathcal{O}(f(n))$
- $\forall c \in \mathbb{N} \forall f : \mathbb{N} \to \mathbb{N} : (f(n))^c \in \mathcal{O}(f(n))$

O-Kalkül - Aufgaben

Beweise oder widerlege:

- $n^2 \in o(n^3)$
- $n^3 \in \Omega(n^2), \quad n \in \Theta(\sqrt{n}), \quad 2^{n+1} \in \mathcal{O}(2^n)$
- $\forall c_1, c_2 \in \mathbb{N}_+ \forall f : \mathbb{N} \to \mathbb{N}_+ : c_1 \cdot f(n) + c_2 \in \mathcal{O}(f(n))$
- $\forall c_1, c_2 \in \mathbb{N}_+ \forall f : \mathbb{N} \to \mathbb{N} \ : c_1 \cdot f(n) + c_2 \in \mathcal{O}(f(n))$
- $\forall c \in \mathbb{N} \forall f : \mathbb{N} \to \mathbb{N} : (f(n))^c \in \mathcal{O}(f(n))$

O-Kalkül - Aufgaben

Beweise oder widerlege:

- $n^2 \in o(n^3)$
- $n^3 \in \Omega(n^2), \quad n \in \Theta(\sqrt{n}), \quad 2^{n+1} \in \mathcal{O}(2^n)$
- $\forall c_1, c_2 \in \mathbb{N}_+ \forall f : \mathbb{N} \to \mathbb{N}_+ : c_1 \cdot f(n) + c_2 \in \mathcal{O}(f(n))$
- $\forall c_1, c_2 \in \mathbb{N}_+ \forall f : \mathbb{N} \to \mathbb{N} \ : c_1 \cdot f(n) + c_2 \in \mathcal{O}(f(n))$
- $\forall c \in \mathbb{N} \forall f : \mathbb{N} \to \mathbb{N} : (f(n))^c \in \mathcal{O}(f(n))$

Logarithmus

Regeln

- $\log(a \cdot b) = \log(a) + \log(b)$
- $\log(a/b) = \log(a) - \log(b)$
- $\log_a(a) = 1$
- $a^{\log_a(b)} = b$

Aufgaben

- $\log(10 \cdot n) \in \mathcal{O}(\log(n)), \quad n^n \in \Theta(2^{n \log(n)})$?

Logarithmus

Regeln

- $\log(a \cdot b) = \log(a) + \log(b)$
- $\log(a/b) = \log(a) - \log(b)$
- $\log_a(a) = 1$
- $a^{\log_a(b)} = b$

Aufgaben

- $\log(10 \cdot n) \in \mathcal{O}(\log(n)), \quad n^n \in \Theta(2^{n \log(n)})$?

Schleifeninvarianten

Schleifeninvariante:

- gilt vor und nach Ausführung des Schleifenkörpers
- kann häufig mit vollst. Induktion bewiesen werden
- Finden der Schleifeninvariante erforder Kreativität/Intuition
  - $\Rightarrow$ das ist meistens der schwierige Teil
- **Wichtig:** für *totale Korrektheit* beweisen, dass Algorithmus/Schleife terminiert

Schleifeninvarianten - Beispiel

```
1 Function f(n : ℕ) : ℕ
2 |   a = 1 : ℕ
3 |   i = 0 : ℕ
4 |   while i < n do
5 |   |   a := a + a
6 |   |   i := i + 1
7 |   end
8 |   return a − 1
```

- Was berechnet der Algorithmus?
- Wie ist die (asymptotische) Laufzeit des Algorithmus?
- Beweise die Korrektheit des Algorithmus mithilfer einer geeigneten Schleifeninvariante (Schleifeninvariante abhängig von i).

Schleifeninvarianten - Beispiel

```
1 Function f(n : ℕ) : ℕ
2 |   a = 1 : ℕ
3 |   i = 0 : ℕ
4 |   while i < n do
5 |   |   a := a + a
6 |   |   i := i + 1
7 |   end
8 |   return a − 1
```

- Was berechnet der Algorithmus?
- Wie ist die (asymptotische) Laufzeit des Algorithmus?
- Beweise die Korrektheit des Algorithmus mithilfer einer geeigneten Schleifeninvariante (Schleifeninvariante abhängig von i).

Schleifeninvarianten - Beispiel

```
1 Function f(n : ℕ) : ℕ
2 |   a = 1 : ℕ
3 |   i = 0 : ℕ
4 |   while i < n do
5 |   |   a := a + a
6 |   |   i := i + 1
7 |   end
8 |   return a − 1
```

- Was berechnet der Algorithmus?
- Wie ist die (asymptotische) Laufzeit des Algorithmus?
- Beweise die Korrektheit des Algorithmus mithilfer einer geeigneten Schleifeninvariante (Schleifeninvariante abhängig von i).

Schleifeninvarianten - Beispiel

```
1 Function f(n : ℕ) : ℕ
2     a = 1 : ℕ
3     i = 0 : ℕ
4     while i < n do
5         invariant a = 2^i
6         a := a + a
7         i := i + 1
8     end
9     return a − 1
```

- Was berechnet der Algorithmus?
- Wie ist die (asymptotische) Laufzeit des Algorithmus?
- Beweise die Korrektheit des Algorithmus mithilfer einer geeigneten Schleifeninvariante (Schleifeninvariante abhängig von i).

Schleifeninvarianten - Beweis

Schleifeninvariante: $a = 2^i$

Beweis:

- **IA.:** Vor 1. Schleifendurchlauf gilt $a = 2^0 = 1$ wegen Initialisierung
- **IV.:** Vor dem j-ten Schleifendurchlauf gelte $a = 2^i$
- **IS.:** Zeile 6: $a := a + a \overset{IV.}{=} 2^i + 2^i = 2^{i+1}$
  Zeile 7: $i := i + 1$
  $\Rightarrow$ Nach j. Schleifendurchlauf und damit vor $j + 1$-ten Schleifendurchlauf gilt $a = 2^i$
- $\Rightarrow$ Die Schleifeninvariante gilt vor/nach jedem Schleifendurchlauf
- Schleife terminiert, da i hochgezählt wird und n endlich

Zusammenfassung

- **Pseudocode:** Beschreibt Algo kompakt und präzise auf abstraker Ebene
- **O-Kalkül:** Hiermit lässt sich die (asymptotische) Laufzeit von Algorithmen gut beschreiben
- **Schleifeninvariante:** Hilfreich um die Korrektheit eines Algorithmus zu beweisen
- Wer sucht noch einen Partner für die Übungsblattabgabe?