Datenstrukturen, Algorithmen und Programmierung 2 (DAP2)

Laufzeitanalyse

Maschinenmodell

- Eine Pseudocode-Instruktion braucht einen Zeitschritt
- Wird eine Instruktion r-mal aufgerufen, werden r Zeitschritte benötigt
- Formales Modell: Random Access Machines (RAM Modell)

Idee

- Ignoriere rechnerabhängige Konstanten
- Betrachte Wachstum von T(n) für $n \to \infty$

„Asymptotische Analyse"

Laufzeitanalyse

Idee (asymptotische Laufzeitanalyse)

- Ignoriere konstante Faktoren
- Betrachte das Verhältnis von Laufzeiten für $n \rightarrow \infty$
- Klassifiziere Laufzeiten durch Angabe von „einfachen Vergleichsfunktionen“

Laufzeitanalyse

O-Notation

- $O(f(n)) = \{g(n) : \exists\ c>0, n_0 > 0$, so dass für alle $n \geq n_0$ gilt $g(n) \leq c \cdot f(n)\}$
- (wobei $f(n), g(n)>0$)

Interpretation

- $g(n) \in O(f(n))$ bedeutet, dass $g(n)$ für $n \to \infty$ höchstens genauso stark wächst wie $f(n)$
- Beim Wachstum ignorieren wir Konstanten

Laufzeitanalyse

Beispiele

- $10n \in O(n)$
- $10n \in O(n^2)$
- $n^2 \notin O(1000n)$
- $O(1000n) = O(n)$

Hierarchie

- $O(\log n) \subseteq O(\log^2 n) \subseteq O(\log^c n) \subseteq O(n^\varepsilon) \subseteq O(\sqrt{n}) \subseteq O(n)$
- $O(n) \subseteq O(n^2) \subseteq O(n^c) \subseteq O(2^n)$
- (für $c \geq 2$ und $0 < \varepsilon \leq 1/2$)

Laufzeitanalyse

Ω-Notation

- $\mathbf{\Omega}(f(n)) = \{g(n) : \exists\ c>0, n_0 >0$, so dass für alle $n \geq n_0$ gilt $g(n) \geq c \cdot f(n)\}$
- (wobei $f(n), g(n)>0$)

Interpretation

- $g(n) \in \mathbf{\Omega}(f(n))$ bedeutet, dass $g(n)$ für $n \rightarrow \infty$ mindestens so stark wächst wie $f(n)$
- Beim Wachstum ignorieren wir Konstanten

Laufzeitanalyse

Beispiele

- $10n \in \Omega(n)$
- $1000n \notin \Omega(n^2)$
- $n^2 \in \Omega(n)$
- $\Omega(1000n) = \Omega(n)$
- $f(n) = \Omega(g(n)) \Leftrightarrow g(n) = O(f(n))$

Laufzeitanalyse

Θ-Notation

- $g(n) \in \Theta(f(n)) \Leftrightarrow g(n)=O(f(n))$ und $g(n)=\Omega(f(n))$

Beispiele

- $1000n \in \Theta(n)$
- $10n^2+1000n \in \Theta(n^2)$
- $n^{1-\sin n} \notin \Theta(n)$

Laufzeitanalyse

o-Notation

- $o(f(n)) = \{g(n): \forall c>0\ \exists n_0>0$, so dass für alle $n \geq n_0$ gilt $c \cdot g(n)<f(n)\}$
- $(f(n), g(n) >0)$

ω-Notation

- $f(n) \in \omega(g(n)) \Leftrightarrow g(n) \in o(f(n))$

Laufzeitanalyse

Beispiele

- $n \in o(n^2)$
- $n \notin o(n)$

Eine weitere Interpretation

- Grob gesprochen sind $O, \Omega, \Theta, o, \omega$ die „asymptotischen Versionen" von $\leq, \geq, =, <, >$ (in dieser Reihenfolge)

Schreibweise

- Wir schreiben häufig $f(n)=O(g(n))$ anstelle von $f(n) \in O(g(n))$

Was ist ein mathematischer Beweis?

Informale Definition

- Ein Beweis ist eine Herleitung einer Aussage aus bereits bewiesenen Aussagen und/oder Grundannahmen (Axiomen).

Korrektheitsbeweise

Was muss ich eigentlich zeigen?

- Häufiges Problem: Was muss man in einem Korrektheitsbeweis beweisen?

Was wissen wir?

- Problembeschreibung definiert zulässige Eingaben und zugehörige (gewünschte) Ausgaben

Korrektheitsbeweise

Wann ist ein Algorithmus korrekt?

- Wir bezeichnen einen Algorithmus für eine vorgegebene Problembeschreibung als korrekt, wenn er für jede Eingabe die in der Problembeschreibung spezifizierte Ausgabe berechnet
- Streng genommen, kann man also nur von Korrektheit sprechen, wenn vorher das angenommene Verhalten des Algorithmus geeignet beschrieben wurde

Beispiel: Sortieren

- Problem: Sortieren
- Eingabe: Folge von n Zahlen $(a_1, \ldots, a_n)$
- Ausgabe: Permutation $(a'_1, \ldots, a'_n)$ von $(a_1, \ldots, a_n)$, so dass $a'_1 \leq a'_2 \leq \ldots \leq a'_n$

Korrektheitsbeweise

Was müssen wir zeigen?

- Für jede gültige Eingabe sortiert unser Algorithmus korrekt

Aber wie? (auf welchen Annahmen können wir aufbauen?)

- Die Grundannahme in der Algorithmik ist, dass ein Pseudocodebefehl gemäß seiner Spezifikation ausgeführt wird
- Z.B.: Die Anweisung $x \leftarrow x + 1$ bewirkt, dass die Variable x um eins erhöht wird

Korrektheitsbeweise

Ein triviales Beispiel

EinfacherAlgorithmus(n)

1. $X \leftarrow 10$
2. $Y \leftarrow n$
3. $X \leftarrow X + Y$
4. ***return*** X

Behauptung

Der Algorithmus gibt den Wert 10+n zurück.

Beweis:

Korrektheitsbeweise

Ein triviales Beispiel

EinfacherAlgorithmus(n)

1. $X \leftarrow 10$
2. $Y \leftarrow n$
3. $X \leftarrow X + Y$
4. **return** X

Behauptung

Der Algorithmus gibt den Wert 10+n zurück.

Beweis:

Zu Beginn des Algorithmus sind alle Variablen bis auf den Parameter n undefiniert.

Korrektheitsbeweise

Ein triviales Beispiel

EinfacherAlgorithmus(n)

1. $\boxed{X \leftarrow 10}$
2. $Y \leftarrow n$
3. $X \leftarrow X + Y$
4. **return** X

Behauptung

Der Algorithmus gibt den Wert 10+n zurück.

Beweis:

Zu Beginn des Algorithmus sind alle Variablen bis auf den Parameter n undefiniert. Der Befehl in Zeile 1 weist X den Wert 10 zu.

Korrektheitsbeweise

Ein triviales Beispiel

EinfacherAlgorithmus(n)

1. $X \leftarrow 10$
2. $\boxed{Y \leftarrow n}$
3. $X \leftarrow X + Y$
4. **return** X

Behauptung

Der Algorithmus gibt den Wert 10+n zurück.

Beweis:

Zu Beginn des Algorithmus sind alle Variablen bis auf den Parameter n undefiniert. Der Befehl in Zeile 1 weist X den Wert 10 zu. Der Befehl in Zeile 2 weist Y den Wert n zu.

Korrektheitsbeweise

Ein triviales Beispiel

EinfacherAlgorithmus(n)

1. $X \leftarrow 10$
2. $Y \leftarrow n$
3. $\boxed{X \leftarrow X + Y}$
4. **return** X

Behauptung

Der Algorithmus gibt den Wert 10+n zurück.

Beweis:

Zu Beginn des Algorithmus sind alle Variablen bis auf den Parameter n undefiniert. Der Befehl in Zeile 1 weist X den Wert 10 zu. Der Befehl in Zeile 2 weist Y den Wert n zu. Der Befehl in Zeile 3 weist X den Wert X + Y zu. Da X vor der Zuweisung den Wert 10 enthielt und Y den Wert n, wird X auf 10+n gesetzt.

Korrektheitsbeweise

Ein triviales Beispiel

EinfacherAlgorithmus(n)

1. $X \leftarrow 10$
2. $Y \leftarrow n$
3. $X \leftarrow X + Y$
4. **return** X

Behauptung

Der Algorithmus gibt den Wert 10+n zurück.

Beweis:

Zu Beginn des Algorithmus sind alle Variablen bis auf den Parameter n undefiniert. Der Befehl in Zeile 1 weist X den Wert 10 zu. Der Befehl in Zeile 2 weist Y den Wert n zu. Der Befehl in Zeile 3 weist X den Wert X + Y zu. Da X vor der Zuweisung den Wert 10 enthielt und Y den Wert n, wird X auf 10+n gesetzt. Der Befehl in Zeile 4 gibt X zurück. Da X zu diesem Zeitpunkt den Wert 10+n hat, folgt die Behauptung.

Korrektheitsbeweise

Ein triviales Beispiel

EinfacherAlgorithmus(n)

1. $X \leftarrow 10$
2. $Y \leftarrow n$
3. $X \leftarrow X + Y$
4. **return** X

Ein Korrektheitsbeweis vollzieht also das Programm Schritt für Schritt nach.

Behauptung

Der Algorithmus gibt den Wert 10+n zurück.

Beweis:

Zu Beginn des Algorithmus sind alle Variablen bis auf den Parameter n undefiniert. Der Befehl in Zeile 1 weist X den Wert 10 zu. Der Befehl in Zeile 2 weist Y den Wert n zu. Der Befehl in Zeile 3 weist X den Wert X + Y zu. Da X vor der Zuweisung den Wert 10 enthielt und Y den Wert n, wird X auf 10+n gesetzt. Der Befehl in Zeile 4 gibt X zurück. Da X zu diesem Zeitpunkt den Wert 10+n hat, folgt die Behauptung.

Korrektheitsbeweise

Ein erstes nichttriviales Beispiel

Algorithmus Max-Search(Array A)

1. max ← 1
2. **for** j ← 2 **to** length[A] **do**
3.   **if** A[j] > A[max] **then** max ← j
4. **return** max

Problem

Wir wissen nicht, wieviele Durchläufe die **for**-Schleife benötigt. Dies hängt sogar von der Eingabelänge ab.

Korrektheitsbeweise

Ein erstes nichttriviales Beispiel

Algorithmus Max-Search(Array A)

1. max ← 1
2. **for** j ← 2 **to** length[A] **do**
3.   **if** A[j] > A[max] **then** max ← j
4. **return** max

Abhilfe

Wir benötigen eine Aussage, die den Zustand am Ende der Schleife nach einer beliebigen Anzahl Schleifendurchläufe angibt.

Korrektheitsbeweise

Ein erstes nichttriviales Beispiel

Algorithmus Max-Search(Array A)

1. max $\leftarrow$ 1
2. **for** j $\leftarrow$ 2 **to** length[A] **do**
3.   **if** A[j] > A[max] **then** max $\leftarrow$ j
4. **return** max

Definition (Schleifeninvariante)

Eine Schleifeninvariante ist eine i.a. von der Anzahl i der Schleifendurchläufe abhängige Aussage A(i), die zu Beginn des i-ten Schleifendurchlauf gilt. Mit A(1) beziehen wir uns also auf den Zustand zu Beginn des ersten Durchlaufs. Dieser wird auch als Initialisierung bezeichnet.

Korrektheitsbeweise

Ein erstes nichttriviales Beispiel

Algorithmus Max-Search(Array A)

1. max $\leftarrow$ 1
2. **for** j $\leftarrow$ 2 **to** length[A] **do**
3.   **if** A[j] > A[max] **then** max $\leftarrow$ j
4. **return** max

*Schleifeninvariante (Konventionen für **for**-Schleifen)*

Bei einer for-Schleife nehmen wir dabei an, dass bereits am Ende eines Schleifendurchlaufs die Laufvariable erhöht wird. Außerdem nehmen wir an, dass zur Initialisierung die Laufvariable bereits auf ihren Startwert initialisiert wurde.

Korrektheitsbeweise

Ein erstes nichttriviales Beispiel

Algorithmus Max-Search(Array A)

1. max ← 1
2. **for** j ← 2 **to** length[A] **do**
3. **if** A[j] > A[max] **then** max ← j
4. **return** max

*Schleifeninvariante (Konventionen für **for**-Schleifen, Teil 2)*

Da bei for-Schleifen die Anzahl der Durchläufe direkt von der Laufvariable abhängt, können wir eine Schleifeninvariante auch in Abhängigkeit der Laufvariablen formulieren.

Korrektheitsbeweise

Ein erstes nichttriviales Beispiel

Algorithmus Max-Search(Array A)

1. max $\leftarrow$ 1
2. **for** j $\leftarrow$ 2 **to** length[A] **do**
3.   **if** A[j] > A[max] **then** max $\leftarrow$ j
4. **return** max

Schleifeninvariante (Zustand nach Austritt aus der Schleife)

Eine Schleife wird beendet, wenn beim Überprüfen der Schleifenbedingung eine Verletzung derselben festgestellt wird. Der danach angenommene Zustand des Algorithmus wird als Austrittszustand bezeichnet und sollte i.a. nicht direkt von der Anzahl der Schleifendurchläufe abhängen.

Korrektheitsbeweise

Ein erstes nichttriviales Beispiel

Algorithmus Max-Search(Array A)

1. max $\leftarrow$ 1
2. **for** j $\leftarrow$ 2 **to** length[A] **do**
3.   **if** A[j] > A[max] **then** max $\leftarrow$ j
4. **return** max

Lemma 1

Die **for**-Schleife in Algorithmus Max-Search erfüllt folgende Schleifeninvariante:

(Inv.) A[max] ist ein größtes Element aus A[1..j-1].

Korrektheitsbeweise

Lemma 1

Die **for**-Schleife in Algorithmus Max-Search erfüllt folgende Schleifeninvariante:

(Inv.) A[max] ist ein größtes Element aus A[1..j-1].

Korrektheitsbeweise

Lemma 1

Die **for**-Schleife in Algorithmus Max-Search erfüllt folgende Schleifeninvariante:

(Inv.) A[max] ist ein größtes Element aus A[1..j-1].

<u>Beweis:</u>

Der Befehl in Zeile 1 des Algorithmus setzt max auf 1. Wir zeigen per Induktion über die Laufvariable j, dass (Inv.) erfüllt ist.

Korrektheitsbeweise

Lemma 1

Die **for**-Schleife in Algorithmus Max-Search erfüllt folgende Schleifeninvariante:

(Inv.) A[max] ist ein größtes Element aus A[1..j-1].

<u>Beweis:</u>

Der Befehl in Zeile 1 des Algorithmus setzt max auf 1. Wir zeigen per Induktion über die Laufvariable j, dass (Inv.) erfüllt ist.

(I.A.) Zur Initialisierung der Schleife ist max=1 und j=2. Außerdem enthält A[1..1] nur ein Element, nämlich A[1]. Da A[max] = A[1] ist, ist A[max] ein größtes Element aus A[1..1]. Daher gilt die Invariante zur Initialisierung.

Korrektheitsbeweise

Lemma 1

Die **for**-Schleife in Algorithmus Max-Search erfüllt folgende Schleifeninvariante:

(Inv.) A[max] ist ein größtes Element aus A[1..j-1].

Beweis:

Der Befehl in Zeile 1 des Algorithmus setzt max auf 1. Wir zeigen per Induktion über die Laufvariable j, dass (Inv.) erfüllt ist.

(I.A.) Zur Initialisierung der Schleife ist max=1 und j=2. Außerdem enthält A[1..1] nur ein Element, nämlich A[1]. Da A[max] = A[1] ist, ist A[max] ein größtes Element aus A[1..1]. Daher gilt die Invariante zur Initialisierung.

Korrektheitsbeweise

Lemma 1

Die **for**-Schleife in Algorithmus Max-Search erfüllt folgende Schleifeninvariante:

(Inv.) A[max] ist ein größtes Element aus A[1..j-1].

Beweis(fortgesetzt):

(I.V.) Sei die Invariante erfüllt für $j=j_0 < \text{length}[A]+1$.

Korrektheitsbeweise

Lemma 1

Die **for**-Schleife in Algorithmus Max-Search erfüllt folgende Schleifeninvariante:

(Inv.) A[max] ist ein größtes Element aus A[1..j-1].

Beweis(fortgesetzt):

(I.V.) Sei die Invariante erfüllt für $j=j_0 < \text{length}[A]+1$.

(I.S.) Zu zeigen: Die Invariante ist erfüllt für j +1. (j-> j+1)

Korrektheitsbeweise

Lemma 1

Die **for**-Schleife in Algorithmus Max-Search erfüllt folgende Schleifeninvariante:

(Inv.) A[max] ist ein größtes Element aus A[1..j-1].

<u>Beweis(fortgesetzt):</u>

(I.V.) Sei die Invariante erfüllt für $j=j_0$ < length[A]+1.

(I.S.) Zu zeigen: Die Invariante ist erfüllt für j +1. (j-> j+1)

Wir betrachten den Durchlauf der Schleife mit Laufvariable $j=j_0$.

Korrektheitsbeweise

Lemma 1

Die **for**-Schleife in Algorithmus Max-Search erfüllt folgende Schleifeninvariante:

(Inv.) A[max] ist ein größtes Element aus A[1..j-1].

Beweis(fortgesetzt):

(I.V.) Sei die Invariante erfüllt für $j=j_0$ < length[A]+1.

(I.S.) Zu zeigen: Die Invariante ist erfüllt für j +1. (j-> j+1)

Wir betrachten den Durchlauf der Schleife mit Laufvariable $j=j_0$.

Falls A[j] ≤ A[max] ist, so wird die **then**-Anweisung nicht ausgeführt. Dann ist A[max] auch größtes Element aus A[1..j]. Am Ende der Schleife wird j um 1 erhöht. Somit gilt die Invariante auch für j +1.

Korrektheitsbeweise

Lemma 1

Die **for**-Schleife in Algorithmus Max-Search erfüllt folgende Schleifeninvariante:

(Inv.) A[max] ist ein größtes Element aus A[1..j-1].

Beweis(fortgesetzt):

(I.V.) Sei die Invariante erfüllt für $j=j_0 < \text{length}[A]+1$.

(I.S.) Zu zeigen: Die Invariante ist erfüllt für j +1. (j-> j+1)

Wir betrachten den Durchlauf der Schleife mit Laufvariable $j=j_0$.

Falls A[j] ≤ A[max] ist, so wird die **then**-Anweisung nicht ausgeführt. Dann ist A[max] auch größtes Element aus A[1..j]. Am Ende der Schleife wird j um 1 erhöht. Somit gilt die Invariante auch für j +1.

Korrektheitsbeweise

Lemma 1

Die **for**-Schleife in Algorithmus Max-Search erfüllt folgende Schleifeninvariante:

(Inv.) A[max] ist ein größtes Element aus A[1..j-1].

Beweis(fortgesetzt):

Falls A[j] > A[max] ist, so ist nach I.V. A[j] größer als das größte Element aus A[1...j-1] und somit das größte Element aus A[1..j]. In der **then**-Anweisung wird max=j gesetzt. Damit ist A[max] das größte Element aus A[1..j]. Am Ende der Schleife wird j um 1 erhöht. Damit gilt die Invariante auch für j +1.

Korrektheitsbeweise

Lemma 1

Die **for**-Schleife in Algorithmus Max-Search erfüllt folgende Schleifeninvariante:

(Inv.) A[max] ist ein größtes Element aus A[1..j-1].

Beweis(fortgesetzt):

Falls A[j] > A[max] ist, so ist nach I.V. A[j] größer als das größte Element aus A[1...j-1] und somit das größte Element aus A[1..j]. In der **then**-Anweisung wird max=j gesetzt. Damit ist A[max] das größte Element aus A[1..j]. Am Ende der Schleife wird j um 1 erhöht. Damit gilt die Invariante auch für j +1.

Nach dem Prinzip der vollständigen Induktion ist somit die Invariante vor jedem Schleifendurchlauf und vor dem Schleifenaustritt erfüllt. Somit gilt die Invariante.

Korrektheitsbeweise

Satz 2

Algorithmus Max-Search berechnet den Index eines größten Element aus einem Feld A.

Beweis

Der Schleifenaustritt aus der for-Schleife (Zeile 2) erfolgt für j=length[A]+1. Nach Lemma 1 gilt die Invariante insbesondere beim Schleifenaustritt und somit, dass A[max] ein größtes Element aus A[1..length[A]] ist. Der **return**-Befehl gibt mit max daher den Index eines größten Elementes aus A zurück.

Korrektheitsbeweise

Ein erstes nichttriviales Beispiel

Algorithmus Max-Search(Array A)

1. max $\leftarrow$ 1
2. **for** j $\leftarrow$ 2 **to** length[A] **do**
3. **if** A[j] > A[max] **then** max $\leftarrow$ j
4. **return** max

Invarianten im Praktikum

Wir werden im Praktikum Invarianten zur Kommentierung von Schleifen nutzen. Diese kann man auch mit Hilfe von „Assertions" zur Laufzeit überprüfen (dazu mehr im Praktikum).

Korrektheitsbeweise

Notation: Invarianten

Algorithmus Max-Search(Array A)

1. max $\leftarrow$ 1
2. **for** j $\leftarrow$ 2 **to** length[A] **do**
3. **if** A[j] > A[max] **then** max $\leftarrow$ j
4. **return** max

- Kommentare (Invariante):
- Initialisierung: max=1, j=2, A[max] ist
- Maximum von A[1..1].
- Invariante: A[max] ist Maximum von A[1..j-1]
- Austritt: A[max] ist Maximum von
- A[1..length[A]]