



Datenstrukturen, Algorithmen und Programmierung 2 (DAP2)

1. Teil der Vorlesung – Grundlagen der Algorithmenanalyse

Inhalt

- Wie beschreibt man einen Algorithmus?
- Wie beweist man die Korrektheit eines Algorithmus?
- Rechenmodell
- Laufzeitanalyse

Organisatorisches

Anmeldung Praktikum

- Die Anmeldung zum Praktikum wird bis Montag 10 Uhr verlängert
- Bitte achten Sie nach Festlegung der (normalen) Übungen auf Überschneidungen mit angegebenen Terminen im Praktikum (Dies macht AsSESS nicht automatisch!)

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

Beispiel: Sortieren

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

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

Ein Korrektheitsbeweis vollzieht also das Programm Schritt für Schritt nach.

Behauptung

Der Algorithmus gibt der 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. $\text{max} \leftarrow 1$
2. **for** $j \leftarrow 2$ **to** $\text{length}[A]$ **do**
3. **if** $A[j] > A[\text{max}]$ **then** $\text{max} \leftarrow j$
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. $\text{max} \leftarrow 1$
2. **for** $j \leftarrow 2$ **to** $\text{length}[A]$ **do**
3. **if** $A[j] > A[\text{max}]$ **then** $\text{max} \leftarrow j$
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. $\text{max} \leftarrow 1$
2. **for** $j \leftarrow 2$ **to** $\text{length}[A]$ **do**
3. **if** $A[j] > A[\text{max}]$ **then** $\text{max} \leftarrow j$
4. **return** max

Definition (Schleifeninvariante)

Eine Schleifeninvariante ist eine i.a. von der Anzahl i der Schleifendurchläufe abhängige Aussage $A(i)$, die am Ende jedes Schleifendurchlaufs gilt. Mit $A(0)$ beziehen wir uns auf den Zustand vor dem ersten Durchlauf.

Korrektheitsbeweise

Ein erstes nichttriviales Beispiel

Algorithmus Max-Search(Array A)

1. $\text{max} \leftarrow 1$
2. **for** $j \leftarrow 2$ **to** $\text{length}[A]$ **do**
3. **if** $A[j] > A[\text{max}]$ **then** $\text{max} \leftarrow j$
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 nach dem 0-ten Durchlauf die Laufvariable bereits auf ihren Startwert initialisiert wurde.

Korrektheitsbeweise

Ein erstes nichttriviales Beispiel

Algorithmus Max-Search(Array A)

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

Lemma 1

Sei $0 \leq i \leq \text{length}[A]-1$ die Anzahl Schleifendurchläufe der **for**-Schleife in Algorithmus Max-Search. Dann gilt folgende Schleifeninvariante

(Inv.) $A[\text{max}]$ ist ein größtes Element aus $A[1..i+1]$ und $j = i+2$.

Korrektheitsbeweise

Lemma 1

Sei $0 \leq i \leq \text{length}[A]-1$ die Anzahl Schleifendurchläufe der **for**-Schleife in Algorithmus Max-Search. Dann gilt folgende Schleifeninvariante

(Inv.) $A[\text{max}]$ ist ein größtes Element aus $A[1..i+1]$ und $j = i+2$.

Beweis:

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

(I.A.) Nach $i=0$ Schleifendurchläufen (also zu Beginn der Schleife) ist $\text{max}=1$. Außerdem enthält $A[1..1]$ nur ein Element, nämlich $A[1]$. Da $A[\text{max}] = A[1]$ ist, ist $A[\text{max}]$ ein größtes Element aus $A[1..1]$. Außerdem ist $j=2$.

Korrektheitsbeweise

Lemma 1

Sei $0 \leq i \leq \text{length}[A]-1$ die Anzahl Schleifendurchläufe der **for**-Schleife in Algorithmus Max-Search. Dann gilt folgende Schleifeninvariante

(Inv.) $A[\text{max}]$ ist ein größtes Element aus $A[1..i+1]$ und $j = i+2$.

Beweis(fortgesetzt):

(I.V.) Sei nach $i \leq \text{length}[A]-1$ Schleifendurchläufen $A[\text{max}]$ ein größtes Element aus $A[1..i+1]$ und $j = i+2$.

(I.S.) Sei $i+1 \leq \text{length}[A]-1$; ansonsten ist nichts zu zeigen. Nach I.V. gilt nach i Durchläufen zu Beginn der Schleife $j = i+2$. Wir betrachten den $(i+1)$ -sten Durchlauf. Falls $A[j] \leq A[\text{max}]$ ist, so wird die **then**-Anweisung nicht ausgeführt. Dann ist $A[\text{max}]$ größtes Element aus $A[1..i+2]$. Am Ende der Schleife wird j um eins erhöht und es gilt $j = i+3$. Damit gilt das Lemma in diesem Fall auch für $i+1$.

Korrektheitsbeweise

Lemma 1

Sei $0 \leq i \leq \text{length}[A]-1$ die Anzahl Schleifendurchläufe der **for**-Schleife in Algorithmus Max-Search. Dann gilt folgende Schleifeninvariante

(Inv.) $A[\text{max}]$ ist ein größtes Element aus $A[1..i+1]$ und $j = i+2$.

Beweis(fortgesetzt):

Falls $A[j] > A[\text{max}]$ ist, so ist nach I.V. $A[j]$ größer als das größte Element aus $A[1..i+1]$ und somit das größte Element aus $A[1..i+2]$. In der **then**-Anweisung wird $\text{max}=j$ gesetzt. Damit ist $A[\text{max}]$ das größte Element aus $A[1..i+2]$. Am Ende der Schleife wird j um eins erhöht und es gilt $j = i+3$. Damit gilt das Lemma in diesem Fall auch für $i+1$.

Korrektheitsbeweise

Satz 2

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

Beweis

Nach dem Lemma 1 gilt insbesondere nach $\text{length}[A]-1$ Durchläufen der **for**-Schleife, dass $A[\text{max}]$ ein größtes Element aus $A[1..\text{length}[A]]$ ist und das $j=\text{length}[A]-1+2 = \text{length}[A]+1$ ist. Damit wird die Schleife nicht noch ein weiteres mal durchlaufen. Der **return**-Befehl gibt max und somit den Index eines größten Elementes aus A zurück.

Insertion Sort

InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}[A]$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

Beispiel (siehe Vollversion)

8	15	3	14	7	6	18	19
---	----	---	----	---	---	----	----

Insertion Sort

InsertionSort(Array A)

```
1.  for j ← 2 to length[A] do
2.    key ← A[j]
3.    i ← j-1
4.    while i>0 and A[i]>key do
5.      A[i+1] ← A[i]
6.      i ← i-1
7.    A[i+1] ← key
```

➤ Eingabegröße n

➤ length[A] = n

➤ verschiebe alle Elemente aus

➤ A[1...j-1], die größer als key

➤ sind eine Stelle nach rechts

➤ Speichere key in Lücke

Lemma 3

Ist A[1..j-1] bereits sortiert, so fügt der unterlegte Block (Zeilen 2-7 des Algorithmus) A[j] korrekt in die sortierte Folge A[1..j-1] ein. Die daraus resultierende Folge steht am Ende in A[1..j].

Korrektheit - Insertion Sort

Lemma 3

Ist $A[1..j-1]$ bereits sortiert, so fügt der unterlegte Block (Zeilen 2-7 des Algorithmus) $A[j]$ korrekt in die sortierte Folge $A[1..j-1]$ ein. Die daraus resultierende Folge steht am Ende in $A[1..j]$.

Beweis:

Wir betrachten Zeile 2-7 des Algorithmus und nehmen an, dass $A[1..j-1]$ bereits sortiert ist. In Zeile 2 wird $A[j]$ in `key` abgespeichert und in Zeile 3 wird `i` auf $j-1$ gesetzt. Sei nun A' das Feld A zum Zeitpunkt vor dem ersten Eintritt in die **while**-Schleife. Außerdem sei i^* der größte Index $i \in \{1, \dots, j-1\}$ mit $A[i] \leq \text{key}$ und $i^* = 0$, falls kein solcher Index existiert. Damit bricht die **while**-Schleife genau dann ab, wenn $i = i^*$ ist. Wir zeigen folgende Invariante. Für $0 \leq k \leq j - i^* - 1$ gilt:

- (Inv.) Am Ende des k -ten Durchlaufs der **while**-Schleife gilt
- (a) $A[1..j-k] = A'[1..j-k]$
 - (b) $A[j-k+1..j] = A'[j-k..j-1]$
 - (c) $\text{key} \leq A[j-k]$ und $i = j-1-k$

Korrektheit - Insertion Sort

Beweis (fortgesetzt):

Wir zeigen folgende Invariante. Für $0 \leq k \leq j-i^*-1$ gilt:

(Inv.) Am Ende des k -ten Durchlaufs der **while**-Schleife gilt

(a) $A[1..j-k] = A'[1..j-k]$

(b) $A[j-k+1..j] = A'[j-k..j-1]$

(c) $key \leq A[j-k]$ und $i=j-1-k$

Wir zeigen die Invariante per Induktion über die Anzahl Schleifendurchläufe k .

(I.A.) Für $k=0$ ist $A[1..j] = A'[1..j]$. Außerdem sind $A[j+1..j]$ und $A'[j..j-1]$

beide leer. Weiterhin ist $i=j-1$ und $key=A[j]$. Somit gilt die Invariante.

(I.V.) Die Invariante gelte für festes k , $0 \leq k \leq j-i^*-1$.

(I.S.) Sei $k+1 \leq j-i^*-1$. Nach I.V. gilt die Invariante nach k Durchläufen. In Zeile 5 wird nun $A[i+1]=A[j-k]$ auf den Wert aus $A[i]=A[j-k-1]$ gesetzt. Dieser ist nach I.V. gleich $A'[j-k-1]$. Da sonst keine Änderung an A vorgenommen wird, gelten somit (a) und (b) auch für $k+1$.

Korrektheit - Insertion Sort

Beweis (fortgesetzt):

Wir zeigen folgende Invariante. Für $0 \leq k \leq j - i^* - 1$ gilt:

(Inv.) Am Ende des k -ten Durchlaufs der **while**-Schleife gilt

- (a) $A[1..j-k] = A'[1..j-k]$
- (b) $A[j-k+1..j] = A'[j-k..j-1]$
- (c) $key \leq A[j-k]$ und $i = j - 1 - k$

(I.S.) Sei $k+1 \leq j - i^* - 1$. Nach I.V. gilt die Invariante nach k Durchläufen. In Zeile 5 wird nun $A[i+1] = A[j-k]$ auf den Wert aus $A[i] = A[j-k-1]$ gesetzt. Dieser ist nach I.V. gleich $A'[j-k-1]$. Da sonst keine Änderung an A vorgenommen wird, gelten somit (a) und (b) auch für $k+1$.

Außerdem gilt $key < A[j-(k+1)]$, da $k+1 \leq j - i^* - 1$ und somit $j-(k+1) > i^*$ gilt.

Nach der Wahl von i^* gilt somit $key < A[j-(k+1)]$. Weiterhin wird i in Zeile 6 um eins verringert. Somit gilt nach der Ausführung von Zeile 6 $i = j - 1 - (k+1)$. Somit gilt die Invariante auch für $k+1$.

Korrektheit - Insertion Sort

Beweis (fortgesetzt):

Wir zeigen folgende Invariante. Für $0 \leq k \leq j - i^* - 1$ gilt:

(Inv.) Am Ende des k -ten Durchlaufs der **while**-Schleife gilt

- (a) $A[1..j-k] = A'[1..j-k]$
- (b) $A[j-k+1..j] = A'[j-k..j-1]$
- (c) $key \leq A[j-k]$ und $i = j - 1 - k$

Damit wissen wir insbesondere, dass nach $j - i^* - 1$ Durchläufen die Invariante erfüllt ist. Es gilt dann $i = i^*$ und es wird somit kein weiterer Durchlauf durchgeführt. In der letzten Zeile wird nun $A[i+1]$ auf key gesetzt. Nach unserer Invariante gilt $key \leq A[j-k] = A[i^*+1]$. Ist $i^* = 0$, so folgt mit (b), dass die Folge nach dem Einfügen sortiert ist. Ist $i^* > 0$, so gilt nach der Wahl von i^* , dass $A[i^*] \leq key$ ist. Nach (a) gilt $A[1..i^*+1] = A'[1..i^*+1]$, also insbesondere $A[1..i^*] = A'[1..i^*]$. Nach (b) gilt $A[i^*+2..j] = A'[i^*+1..j-1]$. Somit ist nach Einfügen von $key = A'[j]$ an die Stelle i^*+1 das Teilfeld $A[1..j]$ sortiert.

Korrektheit - Insertion Sort

Vereinfachter Algorithmus

InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}[A]$ **do**
2. $A[j]$ wird in $A[1..j-1]$ eingefügt

Lemma 4

Sei $0 \leq k \leq \text{length}[A] - 1$ die Anzahl der Schleifendurchläufe der **for**-Schleife von InsertionSort. Dann gilt folgende Invariante am Ende jedes Schleifendurchlaufs:

(Inv.) $A[1..k+1]$ ist sortiert und $j = k + 2$.

Korrektheit - Insertion Sort

Lemma 4

Sei $0 \leq k \leq \text{length}[A]-1$ die Anzahl der Schleifendurchläufe der **for**-Schleife von InsertionSort. Dann gilt folgende Invariante am Ende jedes Schleifendurchlaufs:

(Inv.) $A[1..k+1]$ ist sortiert und $j=k+2$.

Beweis:

Der Beweis ist per Induktion über die Anzahl der Schleifendurchläufe k .

(I.A.) Nach $k=0$ Schleifendurchläufen ist $A[1..1]$ sortiert und $j=2$.

(I.V.) Das Lemma gilt für ein festes k , $0 \leq k \leq \text{length}[A]-1$.

(I.S.) Sei $k+1 \leq \text{length}[A]-1$. Nach I.V. gilt nach k Durchläufen $A[1..j-1] = A[1..k+1]$. Somit ist $A[1..j-1]$ bereits sortiert. Nach Lemma 3 wird $A[j]$ korrekt eingefügt und somit ist nach Abarbeitung von Zeile 2-7 $A[1..j]$ sortiert. Am Ende der Schleife wird j um eins erhöht. Somit gilt die Invariante auch nach $k+1$ Durchläufen.

Korrektheit - Insertion Sort

Satz 5

InsertionSort sortiert jedes Feld A korrekt.

Beweis:

Die Korrektheit folgt aus Lemma 4 mit $k = \text{length}[A] - 1$.

Korrektheitsbeweise - Rekursionen

Algorithmus Sum(A,n)

1. **if** $n=1$ **then** return $A[1]$
2. **else**
3. $W = \text{Sum}(A, n-1)$
4. **return** $A[n] + W$

Problem

Wir können nicht genau sagen, wie häufig eine Rekursion ausgeführt wird.

Korrektheitsbeweise - Rekursionen

Algorithmus Sum(A,n)

1. **if** $n=1$ **then** return $A[1]$
2. **else**
3. $W = \text{Sum}(A, n-1)$
4. **return** $A[n] + W$

Abhilfe

Rekursion ist das Gegenstück zu Induktion. Man kann daher die Korrektheit leicht per Induktion zeigen.

Korrektheitsbeweise - Rekursionen

Algorithmus Sum(A,n)

1. **if** $n=1$ **then** return $A[1]$
2. **else**
3. $W = \text{Sum}(A, n-1)$
4. **return** $A[n] + W$

Beweis (Induktion über n):

(I.A.) Ist $n=1$, so gibt der Algorithmus in Zeile 1 den Wert $A[1]$ zurück. Dies ist korrekt.

(I.V.) Für fest $n-1 > 0$ berechnet $\text{Sum}(A, n-1)$ die Summe der ersten $n-1$ Einträge von A.

(I.S.) Wir betrachten den Aufruf von $\text{Sum}(A, n)$. Da $n > 1$ ist, wird der else-Fall der ersten if-Anweisung aufgerufen. Dort wird W auf $\text{Sum}(A, n-1)$ gesetzt. Nach I.V. ist dies die Summe der ersten $n-1$ Einträge von A. Nun wird in Zeile 4 $A[n]+W$, also die Summe der ersten n Einträge von A zurückgegeben.

Satz 6

Algorithmus Sum(A,n) berechnet die Summe der ersten n Einträge eines Feldes A.

Zusammenfassung - Korrektheitsbeweise

- Grundannahme der Korrektheitsbeweise ist die korrekte Ausführung der Pseudocode Befehle
- Keine Schleifen: „Schrittweises Nachvollziehen des Programms“
- Schleifen: Korrektheit mittels Invarianten und Induktion
- Rekursion: Korrektheit mittels Induktion