Datenstrukturen und Algorithmen

Vorlesung 1: Algorithmische Komplexität

Joost-Pieter Katoen

Lehrstuhl für Informatik 2
Software Modeling and Verification Group

http://www-i2.rwth-aachen.de/i2/dsal10/

16. April 2010

Übersicht

1. Was sind Algorithmen?
   - Algorithmen und Datenstrukturen
   - Effizienz von Algorithmen
2. Average, Best und Worst Case Laufzeitanalyse
   - Lineare Suche
   - Average-Case Analyse von linearer Suche
3. Organisatorisches
   - Übersicht
   - Übungsbetrieb
   - Prüfung

Übersicht

1. Was sind Algorithmen?
   - Algorithmen und Datenstrukturen
   - Effizienz von Algorithmen

2. Average, Best und Worst Case Laufzeitanalyse
   - Lineare Suche
   - Average-Case Analyse von linearer Suche

3. Organisatorisches
   - Übersicht
   - Übungsbetrieb
   - Prüfung

Algorithmen

Algorithmus

Eine wohldefinierte Rechenvorschrift um ein Problem durch ein Computerprogramm zu lösen.

Algorithmen

Algorithmus

Eine wohldefinierte Rechenvorschrift um ein Problem durch ein Computerprogramm zu lösen.

Beispiel (Algorithmen)

Quicksort, Heapsort, Lineare und Binäre Suche, Graphalgorithmen.

Algorithmen

Algorithmus

Eine wohldefinierte Rechenvorschrift um ein Problem durch ein Computerprogramm zu lösen.

Beispiel (Algorithmen)

Quicksort, Heapsort, Lineare und Binäre Suche, Graphalgorithmen.

Löst ein Rechenproblem, beschrieben durch:

- die zu verarbeitenden Eingaben (Vorbedingung / precondition),
- die erwartete Ausgabe (Nachbedingung / postcondition).

mithilfe einer Folge von Rechenschritten.

Beispiel Rechenproblem: Sortieren

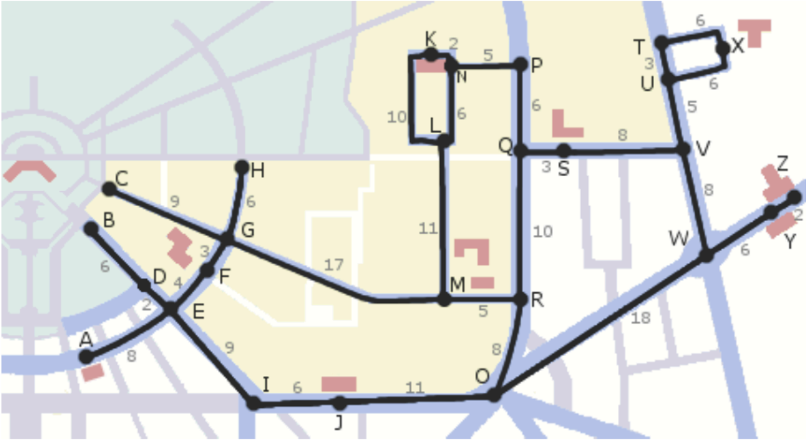
Beispiel

Eingabe: Eine Folge von n natürlichen Zahlen $\langle a_1, a_2, \ldots, a_n \rangle$ mit $a_i \in \mathbb{N}$.

Ausgabe: Eine Permutation (Umordnung) $\langle b_1, b_2, \ldots, b_n \rangle$ der Eingabefolge, sodass $b_1 \leqslant b_2 \leqslant \ldots \leqslant b_n$.

Andere Rechenprobleme: kürzester Weg

Andere Rechenprobleme: kürzester Weg



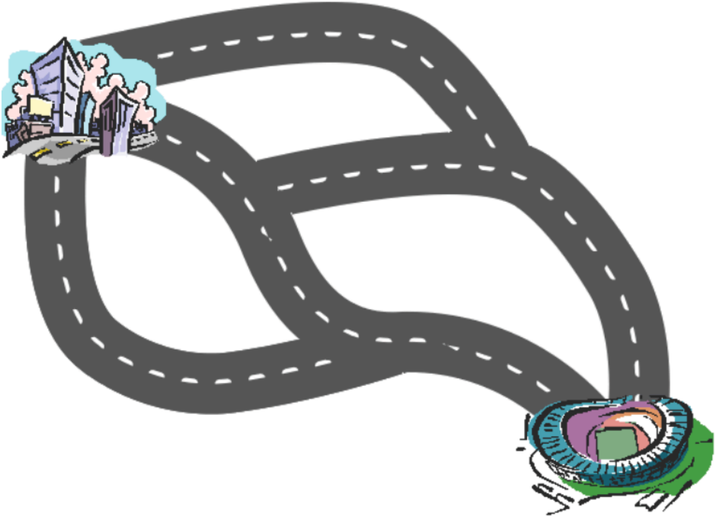
Andere Rechenprobleme: kürzester Weg

Beispiel (kürzester Weg)

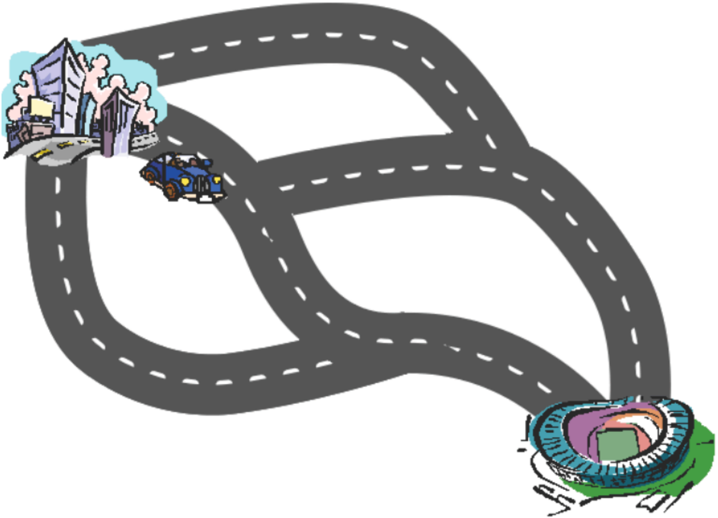
Eingabe:
1. Eine Straßenkarte, auf der der Abstand zwischen jedem Paar benachbarter Kreuzungen eingezeichnet ist,
2. eine Startkreuzung s, und
3. eine Zielkreuzung z.

Ausgabe: Der kürzeste Weg von s nach z.

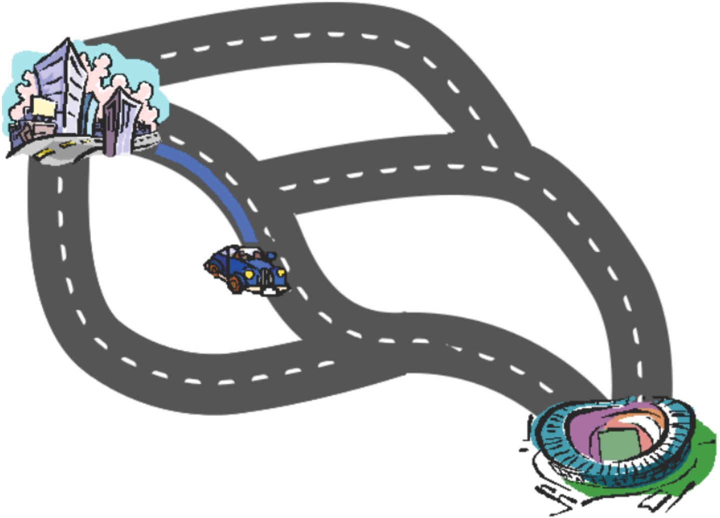
Andere Rechenprobleme: maximale Flüsse



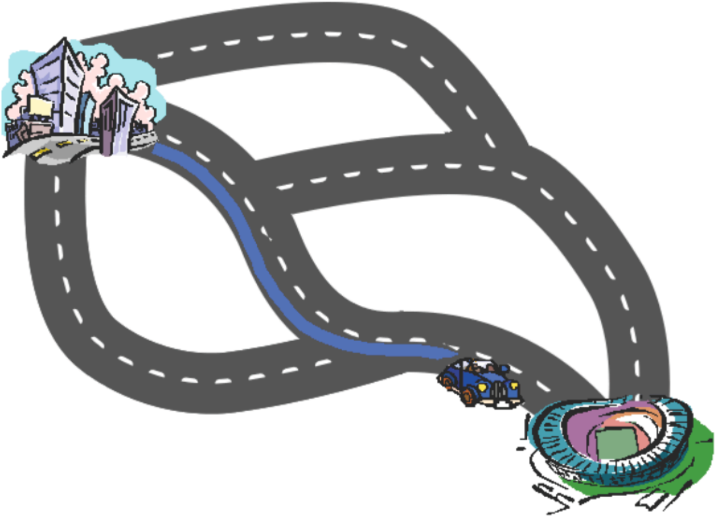
Andere Rechenprobleme: maximale Flüsse



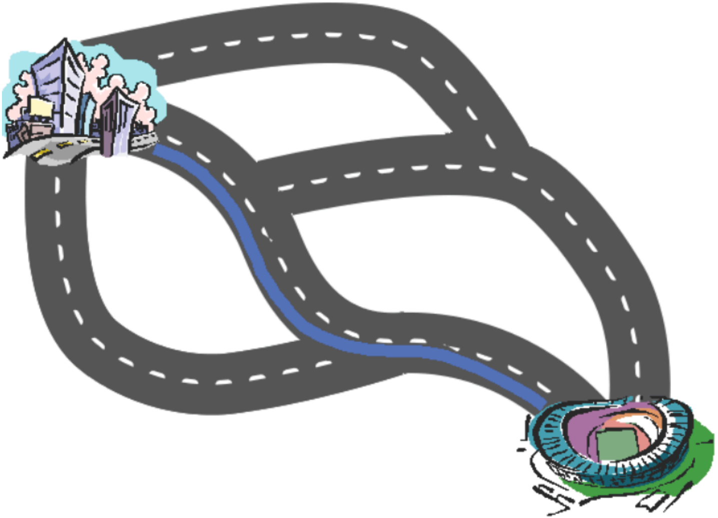
Andere Rechenprobleme: maximale Flüsse



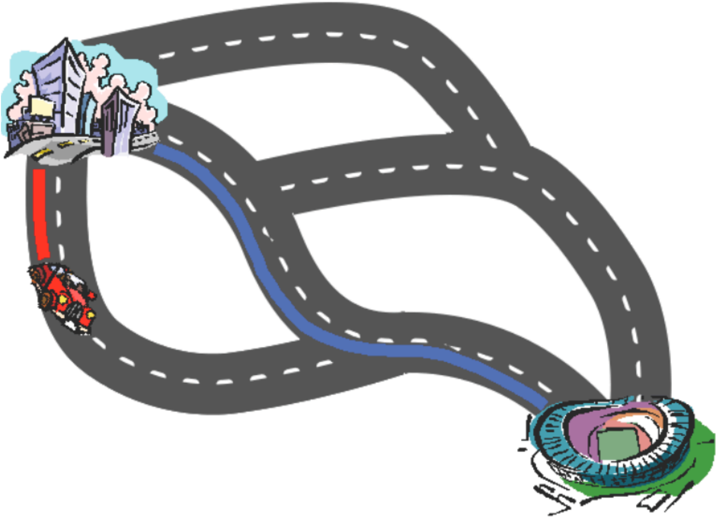
Andere Rechenprobleme: maximale Flüsse



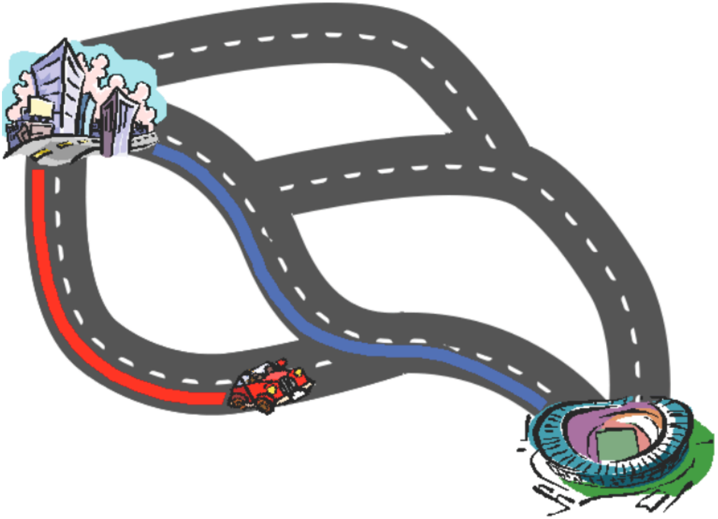
Andere Rechenprobleme: maximale Flüsse



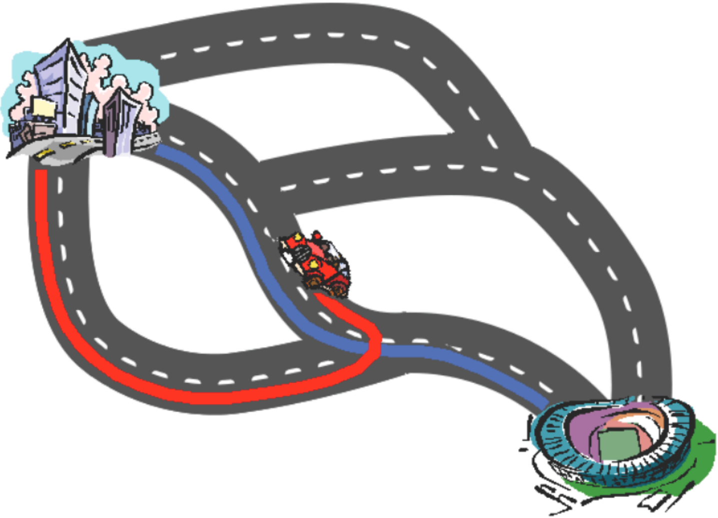
Andere Rechenprobleme: maximale Flüsse



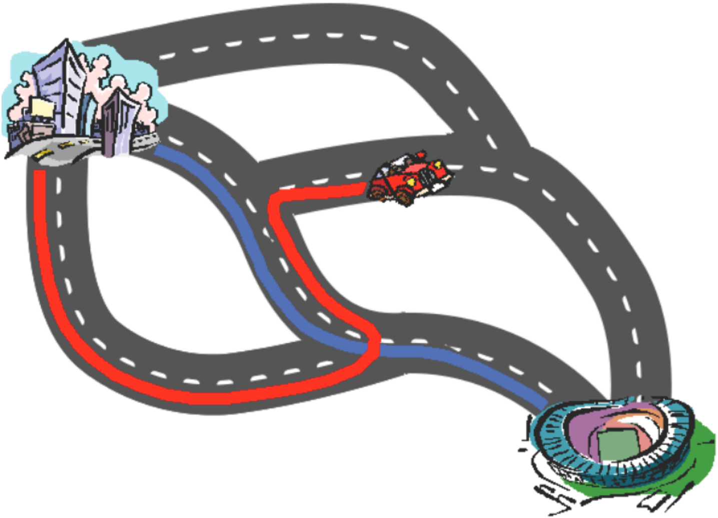
Andere Rechenprobleme: maximale Flüsse



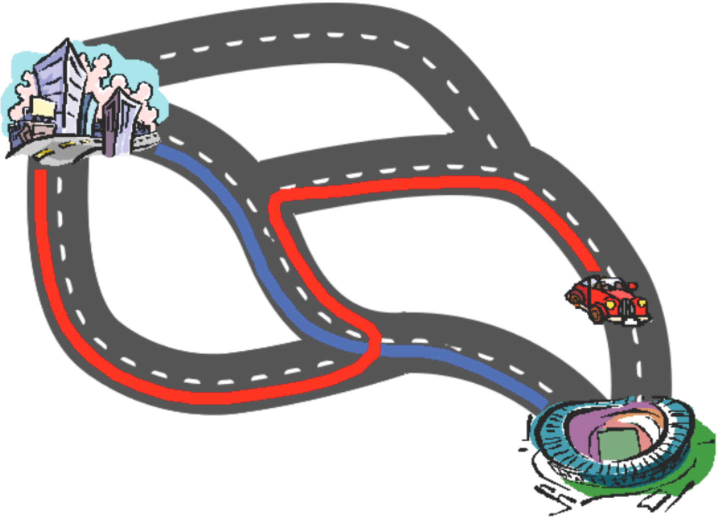
Andere Rechenprobleme: maximale Flüsse



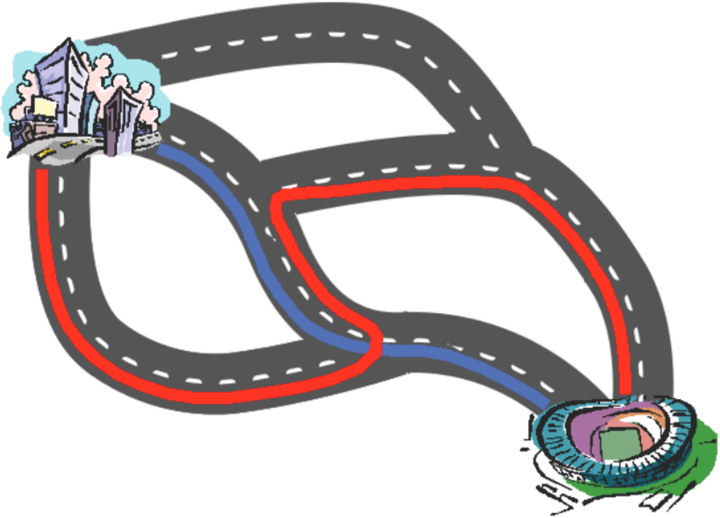
Andere Rechenprobleme: maximale Flüsse



Andere Rechenprobleme: maximale Flüsse



Andere Rechenprobleme: maximale Flüsse



Andere Rechenprobleme: maximale Flüsse

Beispiel (maximale Flüsse)

Eingabe:
1. Eine Straßenkarte, auf der die Kapazität der Straßen eingezeichnet ist,
2. eine Quelle, und
3. eine Senke.

Ausgabe: Die maximale Rate, mit der Material (= Zuschauer) von der Quelle bis zur Senke (= Stadion) transportiert werden kann, ohne die Kapazitätsbeschränkungen der Straßen zu verletzen.

Andere Rechenprobleme: das CD-Brennproblem

Andere Rechenprobleme: das CD-Brennproblem

Betrachte alle Schallplatten von Nina Hagen:

Andere Rechenprobleme: das CD-Brennproblem

Betrachte alle Schallplatten von Nina Hagen:

Wie bekommen wir eine Kompilation ihrer Songs auf einige CDs?

Andere Rechenprobleme: das CD-Brennproblem

Beispiel (CD-Brennproblem)

Eingabe:
1. $N \in \mathbb{N}$ Songs, Song i dauert $0 < n_i \leqslant 80$ Minuten,
2. $k \in \mathbb{N}$ CDs, jeweils mit Kapazität: 80 Minuten.

Andere Rechenprobleme: das CD-Brennproblem

Beispiel (CD-Brennproblem)

Eingabe:
1. $N \in \mathbb{N}$ Songs, Song i dauert $0 < n_i \leqslant 80$ Minuten,
2. $k \in \mathbb{N}$ CDs, jeweils mit Kapazität: 80 Minuten.

Ausgabe: k CDs gefüllt mit einer Auswahl der N Songs, so dass
1. die Songs in chronologische Reihenfolge vorkommen, und

Andere Rechenprobleme: das CD-Brennproblem

Beispiel (CD-Brennproblem)

Eingabe:
1. $N \in \mathbb{N}$ Songs, Song i dauert $0 < n_i \leqslant 80$ Minuten,
2. $k \in \mathbb{N}$ CDs, jeweils mit Kapazität: 80 Minuten.

Ausgabe: k CDs gefüllt mit einer Auswahl der N Songs, so dass
1. die Songs in chronologische Reihenfolge vorkommen, und
2. die totale Dauer der (verschiedenen) ausgewählten Songs maximiert wird,

wobei ein Song komplett auf eine CD gebrannt werden soll.

Algorithmen

Kernpunkte

- Korrektheit: Bei jeder Eingabeinstanz stoppt der Algorithmus mit der korrekten Ausgabe

Algorithmen

Kernpunkte

- Korrektheit: Bei jeder Eingabeinstanz stoppt der Algorithmus mit der korrekten Ausgabe
- Eleganz

Algorithmen

Kernpunkte

- Korrektheit: Bei jeder Eingabeinstanz stoppt der Algorithmus mit der korrekten Ausgabe
- Eleganz
- Effizienz: Wieviel Zeit und Speicherplatz wird benötigt?

Algorithmen

Kernpunkte

- Korrektheit: Bei jeder Eingabeinstanz stoppt der Algorithmus mit der korrekten Ausgabe
- Eleganz
- Effizienz: Wieviel Zeit und Speicherplatz wird benötigt?

Effiziente Algorithmen verwenden effektive Datenstrukturen

Datenstrukturen

Datenstruktur

Ein mathematisches Objekt zur Speicherung von Daten.

Man spricht von einer Struktur, da die Daten in einer bestimmten Art und Weise angeordnet und verknüpft werden, um den Zugriff auf sie und ihre Verwaltung geeignet und effizient zu ermöglichen.

Beispiele (Datenstrukturen)

Array, Baum, Kellerspeicher (stack), Liste,
Warteschlange (queue), Heap, Hashtabelle ...

Effizienz von Algorithmen – Kriterien

Wichtige Kriterien sind (für eine bestimmte Eingabe):

- die benötigte Zeit, Zeitkomplexität
- der benötigte Platz. Platzkomplexität

Zeitkomplexität $\neq$ Platzkomplexität $\neq$ Komplexität des Algorithmus

Ziel

Beurteilung der Effizienz von Algorithmen unabhängig von

- verwendetem Computer, Programmiersprache, Fähigkeiten des Programmierers, usw.

Effizienz von Algorithmen – Elementare Operation

Die Analyse hängt von der Wahl der elementaren Operationen ab, etwa:

- „Vergleich zweier Zahlen“ beim *Sortieren* eines Arrays von Zahlen.
- „Multiplikation zweier Fließkommazahlen“ bei *Matrixmultiplikation*.

Effizienz von Algorithmen – Elementare Operation

Die Analyse hängt von der Wahl der elementaren Operationen ab, etwa:

- „Vergleich zweier Zahlen“ beim *Sortieren* eines Arrays von Zahlen.
- „Multiplikation zweier Fließkommazahlen“ bei *Matrixmultiplikation*.

Elementare Operationen

- Anzahl der elementaren Operationen sollte eine gute Abschätzung für die Anzahl der Gesamtoperationen sein.
- Anzahl der elementaren Operationen bildet die Basis zur Bestimmung der Wachstumsrate der Zeitkomplexität bei immer längeren Eingaben.

Effizienz von Algorithmen – Beispiele

Technologie führt nur zu Verbesserung um einen konstanten Faktor:

Beispiel

Selbst ein Supercomputer kann einen „schlechten“ Algorithmus nicht retten: Für genügend große Eingaben gewinnt *immer* der schnellere Algorithmus auf dem langsameren Computer.

Effizienz von Algorithmen – Beispiele

Technologie führt nur zu Verbesserung um einen konstanten Faktor:

Beispiel

Selbst ein Supercomputer kann einen „schlechten“ Algorithmus nicht retten: Für genügend große Eingaben gewinnt *immer* der schnellere Algorithmus auf dem langsameren Computer.

Beispiel

Typische Laufzeiten (bis auf einen konstanten Faktor) für Eingabelänge n:

1	konstant	$n \cdot \log n$	
$\log n$	logarithmisch	n^2	quadratisch
n	linear	2^n	exponentiell

Zeitkomplexität in der Praxis I

Beispiel (Tatsächliche Laufzeiten)

	Komplexität				
Länge n	$33n$	$46n \log n$	$13n^2$	$3{,}4n^3$	2^n
10	0,00033 s	0,0015 s	0,0013 s	0,0034 s	0,001 s
10^2	0,0033 s	0,03 s	0,13 s	3,4 s	$4 \cdot 10^{16}$ y
10^3	0,033 s	0,45 s	13 s	0,94 h	
10^4	0,33 s	6,1 s	1300 s	39 d	
10^5	3,3 s	1,3 m	1,5 d	108 y	

Benötigte Zeit (s = Sekunde, h = Stunde, d = Tag, y = Jahr)

- Der Einfluss großer konstanter Faktoren nimmt mit wachsendem n ab.

Zeitkomplexität in der Praxis II

Beispiel (Größte lösbare Eingabelänge)

	Komplexität				
Verfügbare Zeit	$33n$	$46n \log n$	$13n^2$	$3{,}4n^3$	2^n
1 s	30 000	2000	280	67	20
1 m	1 800 000	82 000	2170	260	26
1 h	108 000 000	1 180 800	16 818	1009	32
	Größte lösbare Eingabelänge				

Zeitkomplexität in der Praxis II

Beispiel (Größte lösbare Eingabelänge)

	Komplexität				
Verfügbare Zeit	$33n$	$46n \log n$	$13n^2$	$3{,}4n^3$	2^n
1 s	30 000	2000	280	67	20
1 m	1 800 000	82 000	2170	260	26
1 h	108 000 000	1 180 800	16 818	1009	32

Größte lösbare Eingabelänge

- Eine 60-fach längere Eingabe lässt sich nicht durch um den Faktor 60 längere Zeit (oder höhere Geschwindigkeit) bewältigen.

Schnellere Computer. . .

Sei N die größte Eingabelänge, die in fester Zeit gelöst werden kann.

Frage

Wie verhält sich N, wenn wir einen K-mal schnelleren Rechner verwenden?

#Operationen benötigt für Eingabe der Länge n	Größte lösbare Eingabelänge
$\log n$	N^K
n	$K \cdot N$
n^2	$\sqrt{K} \cdot N$
2^n	$N + \log K$

Übersicht

Idee

Wir betrachte einen gegebenen Algorithmus A.

Worst-Case Laufzeit

Die Worst-Case Laufzeit von A ist die von A maximal benötigte Anzahl elementarer Operationen auf einer *beliebigen* Eingabe der Länge n.

Idee

Wir betrachte einen gegebenen Algorithmus A.

Worst-Case Laufzeit

Die Worst-Case Laufzeit von A ist die von A maximal benötigte Anzahl elementarer Operationen auf einer *beliebigen* Eingabe der Länge n.

Best-Case Laufzeit

Die Best-Case Laufzeit von A ist die von A minimal benötigte Anzahl elementarer Operationen auf einer *beliebigen* Eingabe der Länge n.

Idee

Wir betrachte einen gegebenen Algorithmus A.

Worst-Case Laufzeit

Die Worst-Case Laufzeit von A ist die von A maximal benötigte Anzahl elementarer Operationen auf einer *beliebigen* Eingabe der Länge n.

Best-Case Laufzeit

Die Best-Case Laufzeit von A ist die von A minimal benötigte Anzahl elementarer Operationen auf einer *beliebigen* Eingabe der Länge n.

Average-Case Laufzeit

Die Average-Case Laufzeit von A ist die von A durchschnittlich benötigte Anzahl elementarer Operationen auf einer *beliebigen* Eingabe der Länge n.

Idee

Wir betrachte einen gegebenen Algorithmus A.

Worst-Case Laufzeit

Die Worst-Case Laufzeit von A ist die von A maximal benötigte Anzahl elementarer Operationen auf einer *beliebigen* Eingabe der Länge n.

Best-Case Laufzeit

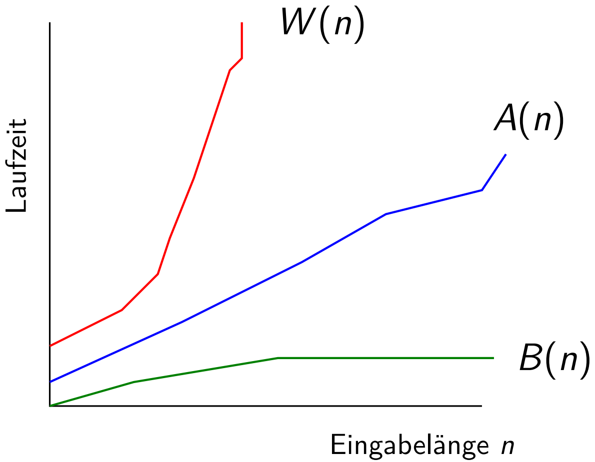
Die Best-Case Laufzeit von A ist die von A minimal benötigte Anzahl elementarer Operationen auf einer *beliebigen* Eingabe der Länge n.

Average-Case Laufzeit

Die Average-Case Laufzeit von A ist die von A durchschnittlich benötigte Anzahl elementarer Operationen auf einer *beliebigen* Eingabe der Länge n.

Alle drei sind Funktionen: Laufzeit in Abhängigkeit von der Eingabelänge!

Beispiel



Formale Definition (I)

Einige hilfreiche Begriffe

$$D_n = \text{Menge aller Eingaben der Länge } n$$

$$t(I) = \text{für Eingabe } I \text{ benötigte Anzahl elementarer Operationen}$$

$$\Pr(I) = \text{Wahrscheinlichkeit, dass Eingabe } I \text{ auftritt}$$

Formale Definition (I)

Einige hilfreiche Begriffe

$$
\begin{aligned}
D_n &= \text{Menge aller Eingaben der Länge } n \\
t(I) &= \text{für Eingabe } I \text{ benötigte Anzahl elementarer Operationen} \\
\Pr(I) &= \text{Wahrscheinlichkeit, dass Eingabe } I \text{ auftritt}
\end{aligned}
$$

Woher kennen wir:

$t(I)$? – Durch Analyse des fraglichen Algorithmus.

$\Pr(I)$? – Erfahrung, Vermutung (z. B. „alle Eingaben treten mit gleicher Wahrscheinlichkeit auf“).

Formale Definition (II)

Worst-Case Laufzeit

Die Worst-Case Laufzeit von A ist die von A maximal benötigte Anzahl elementarer Operationen auf einer *beliebigen* Eingabe der Länge n:

$$W(n) = \max\{\, t(I) \mid I \in D_n \,\}.$$

Formale Definition (II)

Worst-Case Laufzeit

Die Worst-Case Laufzeit von A ist die von A maximal benötigte Anzahl elementarer Operationen auf einer *beliebigen* Eingabe der Länge n:

$$W(n) = \max\{\, t(I) \mid I \in D_n \,\}.$$

Best-Case Laufzeit

Die Best-Case Laufzeit von A ist die von A minimal benötigte Anzahl elementarer Operationen auf einer *beliebigen* Eingabe der Länge n:

$$B(n) = \min\{\, t(I) \mid I \in D_n \,\}.$$

Formale Definition (II)

Average-Case Laufzeit

Die Average-Case Laufzeit von A ist die von A durchschnittlich benötigte Anzahl elementarer Operationen auf einer *beliebigen* Eingabe der Länge n:

$$A(n) = \sum_{I \in D_n} \Pr(I) \cdot t(I)$$

Lineare Suche

Rechenproblem

Eingabe: Array `E` mit `n` Einträgen, sowie das gesuchte Element `K`.

Ausgabe: Ist `K` in `E` enthalten?

Lineare Suche

Rechenproblem

Eingabe: Array `E` mit `n` Einträgen, sowie das gesuchte Element `K`.

Ausgabe: Ist `K` in `E` enthalten?

```
bool linSearch(int E[], int n, int K) {
  for (int index = 0; index < n; index ++) {
    if (E[index] == K) {
      return true; // oder: return index;
    }
  }
  return false; // nicht gefunden
}
```

Lineare Suche – Analyse

Elementare Operation

Vergleich einer ganzen Zahl `K` mit Element `E[index]`.

Lineare Suche – Analyse

Elementare Operation

Vergleich einer ganzen Zahl `K` mit Element `E[index]`.

Menge aller Eingaben

D_n ist die Menge aller Permutationen von n ganzen Zahlen, die ursprünglich aus einer Menge $N > n$ ganzer Zahlen ausgewählt wurden.

Lineare Suche – Analyse

Elementare Operation

Vergleich einer ganzen Zahl `K` mit Element `E[index]`.

Menge aller Eingaben

D_n ist die Menge aller Permutationen von n ganzen Zahlen, die ursprünglich aus einer Menge $N > n$ ganzer Zahlen ausgewählt wurden.

Zeitkomplexität

Lineare Suche – Analyse

Elementare Operation

Vergleich einer ganzen Zahl `K` mit Element `E[index]`.

Menge aller Eingaben

D_n ist die Menge aller Permutationen von n ganzen Zahlen, die ursprünglich aus einer Menge $N > n$ ganzer Zahlen ausgewählt wurden.

Zeitkomplexität

- $W(n) = n$, da n Vergleiche notwendig sind, falls `K` nicht in `E` vorkommt (oder wenn `K == E[n]`).

Lineare Suche – Analyse

Elementare Operation

Vergleich einer ganzen Zahl `K` mit Element `E[index]`.

Menge aller Eingaben

D_n ist die Menge aller Permutationen von n ganzen Zahlen, die ursprünglich aus einer Menge $N > n$ ganzer Zahlen ausgewählt wurden.

Zeitkomplexität

- $W(n) = n$, da n Vergleiche notwendig sind, falls `K` nicht in `E` vorkommt (oder wenn `K == E[n]`).
- $B(n) = 1$, da ein Vergleich ausreicht, wenn `K` gleich `E[1]` ist.

Lineare Suche – Analyse

Elementare Operation

Vergleich einer ganzen Zahl `K` mit Element `E[index]`.

Menge aller Eingaben

D_n ist die Menge aller Permutationen von n ganzen Zahlen, die ursprünglich aus einer Menge $N > n$ ganzer Zahlen ausgewählt wurden.

Zeitkomplexität

- $W(n) = n$, da n Vergleiche notwendig sind, falls `K` nicht in `E` vorkommt (oder wenn `K == E[n]`).
- $B(n) = 1$, da ein Vergleich ausreicht, wenn `K` gleich `E[1]` ist.
- $A(n) \approx \frac{1}{2}n$, da im Schnitt `K` mit etwa der Hälfte der Array `E` verglichen werden muss? – Nein.

Lineare Suche – Average-Case-Analyse (I)

Zwei Szenarien

1. `K` kommt nicht in `E` vor.
2. `K` kommt in `E` vor.

Lineare Suche – Average-Case-Analyse (I)

Zwei Szenarien

1. `K` kommt nicht in `E` vor.
2. `K` kommt in `E` vor.

Zwei Definitionen

1. Sei $A_{K \notin E}(n)$ die Average-Case-Laufzeit für den Fall "`K` nicht in `E`".
2. Sei $A_{K \in E}(n)$ die Average-Case-Laufzeit für den Fall "`K` in `E`".

Lineare Suche – Average-Case-Analyse (I)

Zwei Szenarien

1. `K` kommt nicht in `E` vor.
2. `K` kommt in `E` vor.

Zwei Definitionen

1. Sei $A_{K \notin E}(n)$ die Average-Case-Laufzeit für den Fall "`K` nicht in `E`".
2. Sei $A_{K \in E}(n)$ die Average-Case-Laufzeit für den Fall "`K` in `E`".

$$A(n) = \Pr\{\texttt{K in E}\} \cdot A_{K \in E}(n) + \Pr\{\texttt{K nicht in E}\} \cdot A_{K \notin E}(n)$$

Der Fall "`K` in `E`"

- Nehme an, dass alle Elemente in `E` unterschiedlich sind.

Der Fall "`K` in `E`"

- Nehme an, dass alle Elemente in `E` unterschiedlich sind.
- Damit ist die Wahrscheinlichkeit für `K == E[i]` gleich $\frac{1}{n}$.

Der Fall "`K` in `E`"

- Nehme an, dass alle Elemente in `E` unterschiedlich sind.
- Damit ist die Wahrscheinlichkeit für `K == E[i]` gleich $\frac{1}{n}$.
- Die Anzahl benötigter Vergleiche im Fall `K == E[i]` ist $i+1$.

Der Fall "K in E"

- Nehme an, dass alle Elemente in E unterschiedlich sind.
- Damit ist die Wahrscheinlichkeit für K == E[i] gleich $\frac{1}{n}$.
- Die Anzahl benötigter Vergleiche im Fall K == E[i] ist $i+1$.
- Damit ergibt sich:

$$A_{K \in E}(n) \quad = \quad \sum_{i=0}^{n-1} \Pr\{\texttt{K == E[i]} \,|\, \texttt{K in E}\} \cdot t(\texttt{K == E[i]})$$

Der Fall "K in E"

- Nehme an, dass alle Elemente in E unterschiedlich sind.
- Damit ist die Wahrscheinlichkeit für K == E[i] gleich $\frac{1}{n}$.
- Die Anzahl benötigter Vergleiche im Fall K == E[i] ist $i+1$.
- Damit ergibt sich:

$$\begin{aligned} A_{K \in E}(n) &= \sum_{i=0}^{n-1} \Pr\{\texttt{K == E[i]} \,|\, \texttt{K} \text{ in } \texttt{E}\} \cdot t(\texttt{K == E[i]}) \\ &= \sum_{i=0}^{n-1} \left(\frac{1}{n}\right) \cdot (i+1) \end{aligned}$$

Der Fall "`K` in `E`"

- Nehme an, dass alle Elemente in `E` unterschiedlich sind.
- Damit ist die Wahrscheinlichkeit für `K == E[i]` gleich $\frac{1}{n}$.
- Die Anzahl benötigter Vergleiche im Fall `K == E[i]` ist $i+1$.
- Damit ergibt sich:

$$\begin{aligned}
A_{K \in E}(n) &= \sum_{i=0}^{n-1} \Pr\{\texttt{K == E[i]} \mid \texttt{K in E}\} \cdot t(\texttt{K == E[i]}) \\
&= \sum_{i=0}^{n-1} \left(\frac{1}{n}\right) \cdot (i+1) \\
&= \left(\frac{1}{n}\right) \cdot \sum_{i=0}^{n-1} (i+1)
\end{aligned}$$

Der Fall "K in E"

- Nehme an, dass alle Elemente in E unterschiedlich sind.
- Damit ist die Wahrscheinlichkeit für K == E[i] gleich $\frac{1}{n}$.
- Die Anzahl benötigter Vergleiche im Fall K == E[i] ist $i+1$.
- Damit ergibt sich:

$$
\begin{aligned}
A_{K \in E}(n) &= \sum_{i=0}^{n-1} \Pr\{\texttt{K == E[i]} \mid \texttt{K}\ \text{in}\ \texttt{E}\} \cdot t(\texttt{K == E[i]}) \\
&= \sum_{i=0}^{n-1} \left(\frac{1}{n}\right) \cdot (i+1) \\
&= \left(\frac{1}{n}\right) \cdot \sum_{i=0}^{n-1} (i+1) \\
&= \left(\frac{1}{n}\right) \cdot \frac{n(n+1)}{2}
\end{aligned}
$$

Der Fall "`K` in `E`"

- Nehme an, dass alle Elemente in `E` unterschiedlich sind.
- Damit ist die Wahrscheinlichkeit für `K == E[i]` gleich $\frac{1}{n}$.
- Die Anzahl benötigter Vergleiche im Fall `K == E[i]` ist $i+1$.
- Damit ergibt sich:

$$
\begin{aligned}
A_{K \in E}(n) &= \sum_{i=0}^{n-1} \Pr\{\texttt{K == E[i]} \,|\, \texttt{K}\ \text{in}\ \texttt{E}\} \cdot t(\texttt{K == E[i]}) \\
&= \sum_{i=0}^{n-1} \left(\frac{1}{n}\right) \cdot (i+1) \\
&= \left(\frac{1}{n}\right) \cdot \sum_{i=0}^{n-1} (i+1) \\
&= \left(\frac{1}{n}\right) \cdot \frac{n(n+1)}{2} \\
&= \frac{n+1}{2}.
\end{aligned}
$$

Ableitung

$$A(n) = \Pr\{\texttt{K in E}\} \cdot A_{K \in E}(n) + \Pr\{\texttt{K nicht in E}\} \cdot A_{K \notin E}(n)$$

Ableitung

$$A(n) = \Pr\{\texttt{K in E}\} \cdot A_{K \in E}(n) + \Pr\{\texttt{K nicht in E}\} \cdot A_{K \notin E}(n)$$

$$\Big| \; A_{K \in E}(n) = \frac{n+1}{2}$$

$$= \Pr\{\texttt{K in E}\} \cdot \frac{n+1}{2} + \Pr\{\texttt{K nicht in E}\} \cdot A_{K \notin E}(n)$$

Ableitung

$$A(n) = \Pr\{\texttt{K in E}\} \cdot A_{K \in E}(n) + \Pr\{\texttt{K nicht in E}\} \cdot A_{K \notin E}(n)$$

$$\Big|\; A_{K \in E}(n) = \frac{n+1}{2}$$

$$= \Pr\{\texttt{K in E}\} \cdot \frac{n+1}{2} + \Pr\{\texttt{K nicht in E}\} \cdot A_{K \notin E}(n)$$

$$\Big|\; \Pr\{\text{nicht } B\} = 1 - \Pr\{B\}$$

$$= \Pr\{\texttt{K in E}\} \cdot \frac{n+1}{2} + (1 - \Pr\{\texttt{K in E}\}) \cdot A_{K \notin E}(n)$$

Ableitung

$$A(n) = \Pr\{\texttt{K in E}\} \cdot A_{K \in E}(n) + \Pr\{\texttt{K nicht in E}\} \cdot A_{K \notin E}(n)$$

$$\Big| \; A_{K \in E}(n) = \frac{n+1}{2}$$

$$= \Pr\{\texttt{K in E}\} \cdot \frac{n+1}{2} + \Pr\{\texttt{K nicht in E}\} \cdot A_{K \notin E}(n)$$

$$\Big| \; \Pr\{\text{nicht } B\} = 1 - \Pr\{B\}$$

$$= \Pr\{\texttt{K in E}\} \cdot \frac{n+1}{2} + (1 - \Pr\{\texttt{K in E}\}) \cdot A_{K \notin E}(n)$$

$$\Big| \; A_{K \notin E}(n) = n$$

$$= \Pr\{\texttt{K in E}\} \cdot \frac{n+1}{2} + (1 - \Pr\{\texttt{K in E}\}) \cdot n$$

Ableitung

$$A(n) = \Pr\{\texttt{K in E}\} \cdot A_{K \in E}(n) + \Pr\{\texttt{K nicht in E}\} \cdot A_{K \notin E}(n)$$

$$\left|\; A_{K \in E}(n) = \frac{n+1}{2}\right.$$

$$= \Pr\{\texttt{K in E}\} \cdot \frac{n+1}{2} + \Pr\{\texttt{K nicht in E}\} \cdot A_{K \notin E}(n)$$

$$\left|\; \Pr\{\text{nicht } B\} = 1 - \Pr\{B\}\right.$$

$$= \Pr\{\texttt{K in E}\} \cdot \frac{n+1}{2} + (1 - \Pr\{\texttt{K in E}\}) \cdot A_{K \notin E}(n)$$

$$\left|\; A_{K \notin E}(n) = n\right.$$

$$= \Pr\{\texttt{K in E}\} \cdot \frac{n+1}{2} + (1 - \Pr\{\texttt{K in E}\}) \cdot n$$

$$= n \cdot \left(1 - \frac{1}{2}\Pr\{\texttt{K in E}\}\right) + \frac{1}{2}\Pr\{\texttt{K in E}\}$$

Lineare Suche – Average-Case-Analyse

Endergebnis

Die Average-Case-Zeitkomplexität der linearen Suche ist:

$$A(n) \;=\; n \cdot \left(1 - \frac{1}{2}\,\mathrm{Pr}\{\texttt{K in E}\}\right) + \frac{1}{2}\,\mathrm{Pr}\{\texttt{K in E}\}$$

Lineare Suche – Average-Case-Analyse

Endergebnis

Die Average-Case-Zeitkomplexität der linearen Suche ist:

$$A(n) \; = \; n \cdot \left(1 - \frac{1}{2} \Pr\{\texttt{K} \text{ in } \texttt{E}\}\right) + \frac{1}{2} \Pr\{\texttt{K} \text{ in } \texttt{E}\}$$

Beispiel

Wenn $\Pr\{K \text{ in } E\}$

$= 1$, dann $A(n) = \frac{n+1}{2}$, d. h. etwa 50% von E ist überprüft.

Lineare Suche – Average-Case-Analyse

Endergebnis

Die Average-Case-Zeitkomplexität der linearen Suche ist:

$$A(n) \;=\; n \cdot \left(1 - \frac{1}{2}\,\mathrm{Pr}\{\texttt{K in E}\}\right) + \frac{1}{2}\,\mathrm{Pr}\{\texttt{K in E}\}$$

Beispiel

Wenn $\mathrm{Pr}\{K \text{ in } E\}$

- $= 1$, dann $A(n) = \frac{n+1}{2}$, d. h. etwa 50% von E ist überprüft.
- $= 0$, dann $A(n) = n = W(n)$, d. h. E wird komplett überprüft.

Lineare Suche – Average-Case-Analyse

Endergebnis

Die Average-Case-Zeitkomplexität der linearen Suche ist:

$$A(n) \ = \ n \cdot \left(1 - \frac{1}{2}\,\mathrm{Pr}\{\mathtt{K} \text{ in } \mathtt{E}\}\right) + \frac{1}{2}\,\mathrm{Pr}\{\mathtt{K} \text{ in } \mathtt{E}\}$$

Beispiel

Wenn $\mathrm{Pr}\{K \text{ in } E\}$

$= 1$, dann $A(n) = \frac{n+1}{2}$, d. h. etwa 50% von E ist überprüft.

$= 0$, dann $A(n) = n = W(n)$, d. h. E wird komplett überprüft.

$= \frac{1}{2}$, dann $A(n) = \frac{3 \cdot n}{4} + \frac{1}{4}$, d. h. etwa 75% von E wird überprüft.

Übersicht

Übersicht (Teil I)

1. Algorithmische Komplexität
2. Asymptotische Effizienz
3. Elementare Datenstrukturen
4. Suchen
5. Rekursionsgleichungen
6. Sortieren: in-situ, Mergesort, Heapsort, Quicksort
7. Binäre Suchbäume
8. Rot-schwarz Bäume

Übersicht (Teil II)

1. Hashing
2. Elementare Graphenalgorithmen
3. Minimale Spannbäume
4. Kürzeste Pfadalgorithmen
5. Maximaler Fluss
6. Dynamische Programmierung
7. B-Bäume
8. Algorithmische Geometrie

Literatur

Die Vorlesung orientiert sich im Wesentlichen an diesem Buch:

Thomas H. Cormen, Charles E. Leiserson, Ronald Rivest, Clifford Stein:

Algorithmen - Eine Einführung

R. Oldenbourg Verlag , 2. Auflage 2007.

Wichtige Termine

Vorlesungstermine

Vorlesung: Di. 14:00–15:30, Fr. 14:00–15:30, Großer Hörsaal (Audimax)

Keine Vorlesung am 14. Mai und 1. Juni.

Letzte Vorlesung am 23. Juli.

Wichtige Termine

Vorlesungstermine

Vorlesung: Di. 14:00–15:30, Fr. 14:00–15:30, Großer Hörsaal (Audimax)

Keine Vorlesung am 14. Mai und 1. Juni.

Letzte Vorlesung am 23. Juli.

Frontalübung: Mo. 14:00–15:30, AH IV (Informatikzentrum)

Erste Frontalübung: Mo. 26. April

Übungsbetrieb

Übungsgruppen

- 15 Übungsgruppen: verschiedene Uhrzeiten am Mo.–Mi.
- Spezialübung für Lehramtsstudenten
- 4 Übungsgruppen für Erstsemester
- Koordinatoren: Jonathan Heinen, Sabrina von Styp und Haidi Yue.

Übungsbetrieb

Übungsgruppen

- 15 Übungsgruppen: verschiedene Uhrzeiten am Mo.–Mi.
- Spezialübung für Lehramtsstudenten
- 4 Übungsgruppen für Erstsemester
- Koordinatoren: Jonathan Heinen, Sabrina von Styp und Haidi Yue.

Anmeldung für die Übungsgruppen

Anmeldung zum Übungsbetrieb über CAMPUS-Office bis spätestens

Mittwoch, 21.04., 12 Uhr (Aachener Zeit)

- möglichst viele Prioritäten angeben

Übungsbetrieb

Wichtige Termine

Übungszettel:	Freitags ab 18:00 im Web
Erster Übungszettel:	16. April 2010
Abgabe Übungszettel:	Montags vor 11:00 Uhr im Sammelkasten
Erste Übungsabgabe:	Montag, 26. April 2010
Übungszeiten:	Montag, Dienstag oder Mittwoch
Erste Übungen:	17. Kalenderwoche: 26.–30. April 2010
Frontalübung:	Montags, 14:00–15:30 (AH IV) ab 26. April
Präsenzübung:	Montag, 28. Juni 2010 (13:45–15:30, AH IV)

Prüfung

Die Prüfung ist eine schriftliche Klausur von 120 Minuten.

Zulassungskriterium Klausur

1. Mindestens 50% aller in den Übungen erreichbaren Punkte, und
2. mindestens 50% der in der Präsenzübung erreichbaren Punkte.

CES-Studenten brauchen kein Zulassungskriterium zu erfüllen.

Prüfung

Die Prüfung ist eine schriftliche Klausur von 120 Minuten.

Zulassungskriterium Klausur

1. Mindestens 50% aller in den Übungen erreichbaren Punkte, und
2. mindestens 50% der in der Präsenzübung erreichbaren Punkte.

CES-Studenten brauchen kein Zulassungskriterium zu erfüllen.

Wichtige Termine

Präsenzübung:	Montag, 28. Juni 2010 (13:45–15:30, AH IV)
Klausur:	Dienstag, 10. August 2010 (vormittags)
Wiederholungsklausur:	Montag, 20. September 2010 (nachmittags)

Sonstiges

Mehr Information

- Webseite: `http://moves.rwth-aachen.de/i2/dsal10/`
- Diskussionsforum: `https://www2.elearning.rwth-aachen.de/ss10/10ss-03999/`
- Oder: `http://www.infostudium.de/`
- E-Mail: `dsal@informatik.rwth-aachen.de`