

Datenstrukturen und Algorithmen

Vorlesung 1: Algorithmische Komplexität

Joost-Pieter Katoen

Lehrstuhl für Informatik 2
Software Modeling and Verification Group

<https://moves.rwth-aachen.de/teaching/ss-18/dsal/>

13. April 2018

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) und
- ▶ die erwartete Ausgabe (Nachbedingung / postcondition),

mithilfe einer Folge von Rechenschritten.

Beispiel Rechenproblem: Sortieren

Beispiel

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

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

Andere Rechenprobleme: kürzester Weg



Andere Rechenprobleme: kürzester Weg



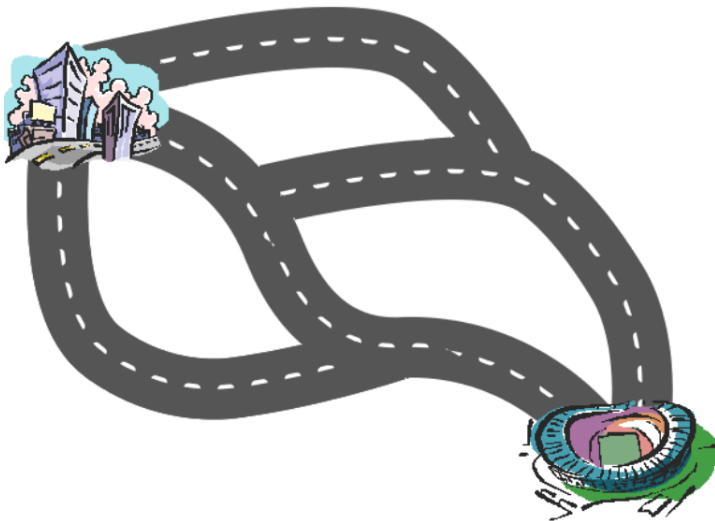
Andere Rechenprobleme: kürzester Weg

Beispiel (kürzester Weg)

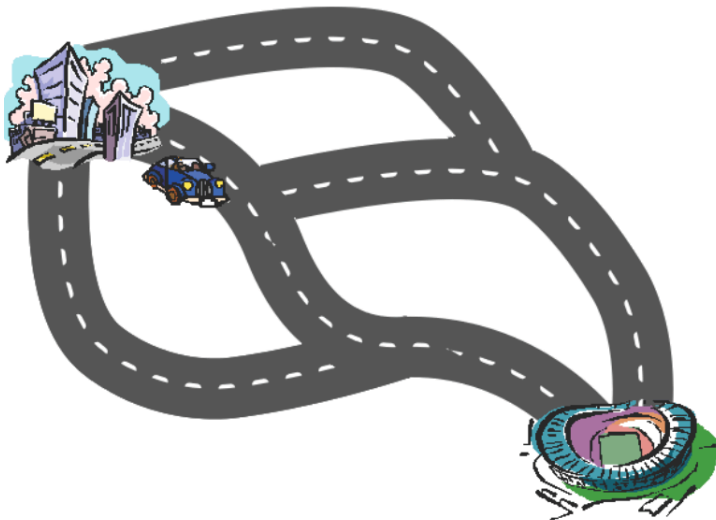
- Eingabe:**
1. Eine Straßenkarte, auf welcher der Abstand zwischen jedem Paar benachbarter Kreuzungen eingezeichnet ist,
 2. eine Startkreuzung s und
 3. eine Zielkreuzung z .

Ausgabe: Ein kürzeste Weg von s nach z .

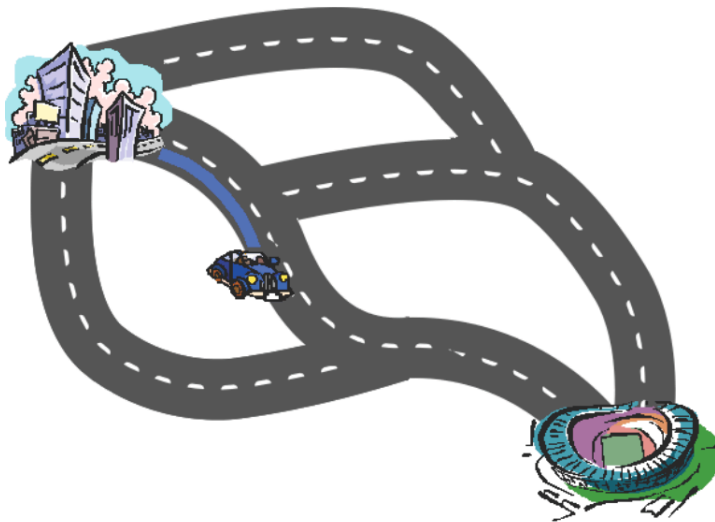
Andere Rechenprobleme: maximale Flüsse



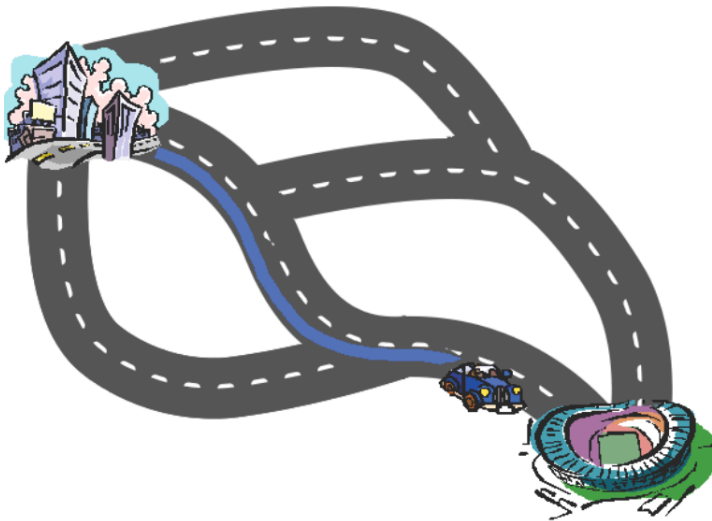
Andere Rechenprobleme: maximale Flüsse



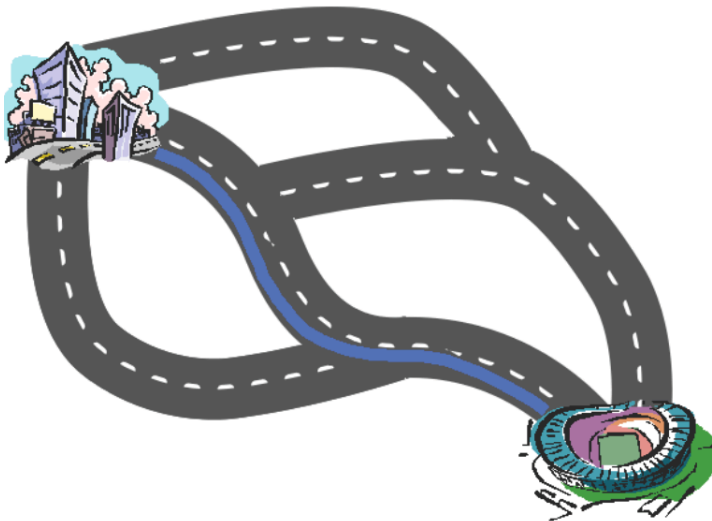
Andere Rechenprobleme: maximale Flüsse



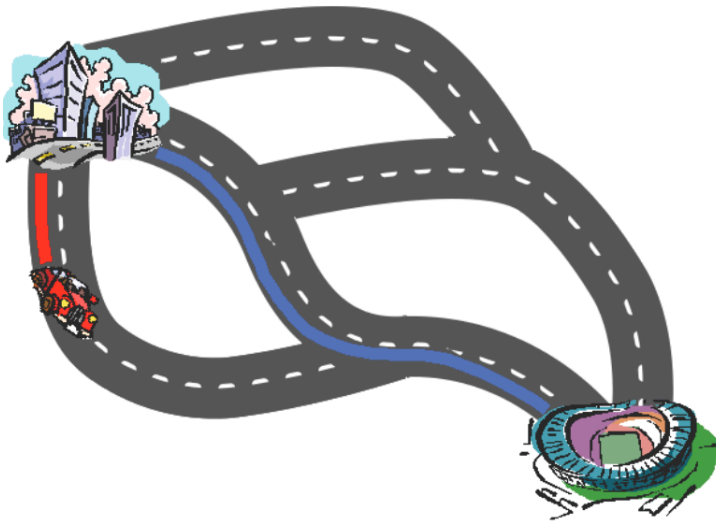
Andere Rechenprobleme: maximale Flüsse



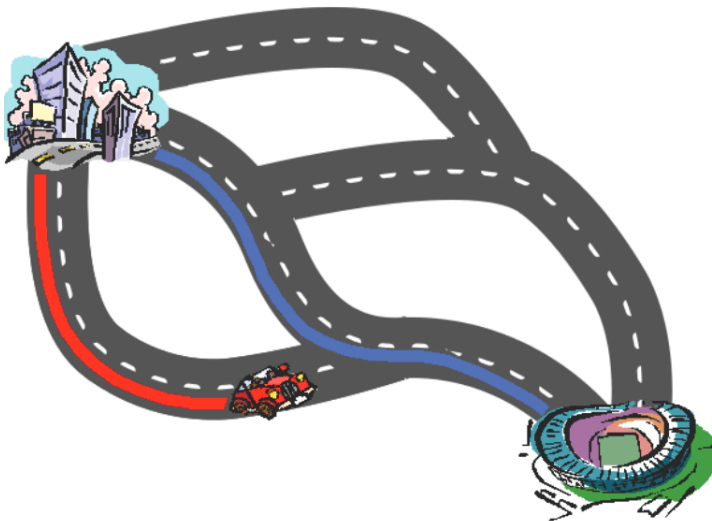
Andere Rechenprobleme: maximale Flüsse



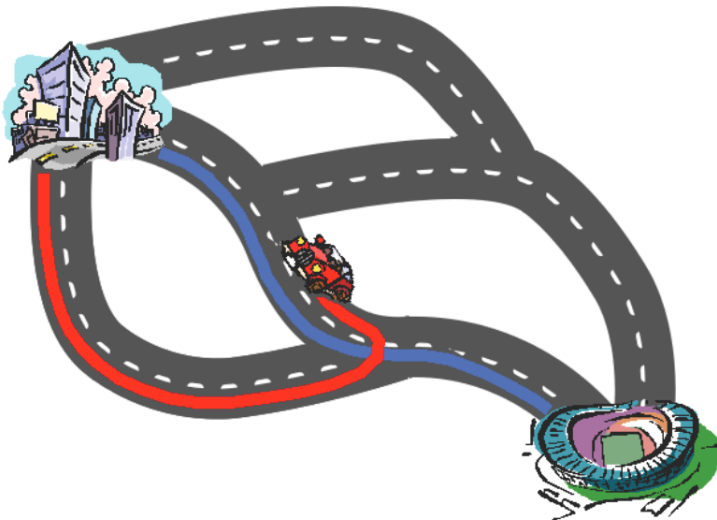
Andere Rechenprobleme: maximale Flüsse



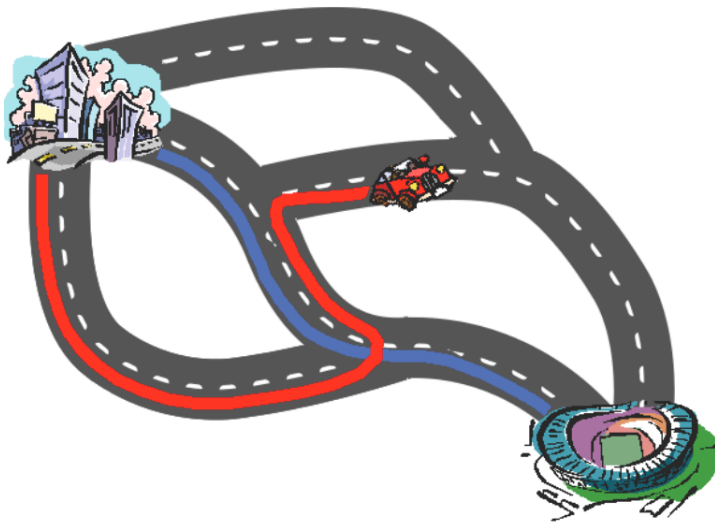
Andere Rechenprobleme: maximale Flüsse



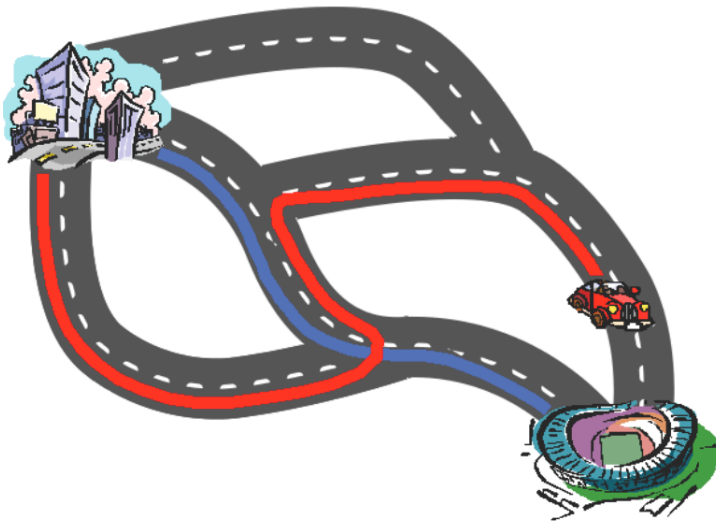
Andere Rechenprobleme: maximale Flüsse



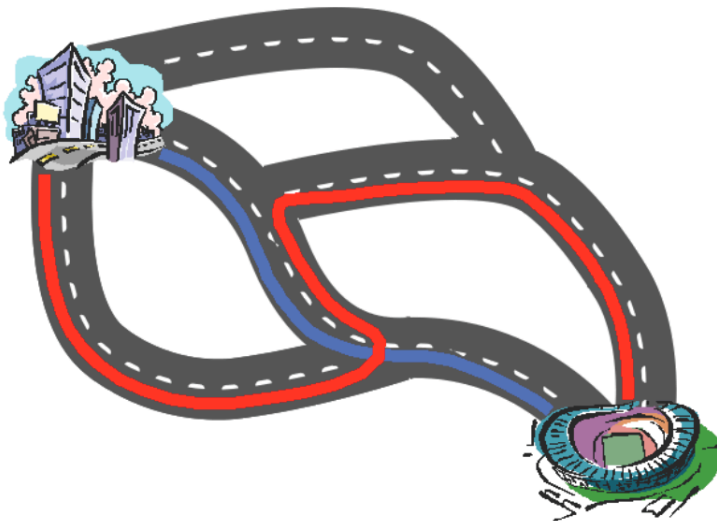
Andere Rechenprobleme: maximale Flüsse



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.

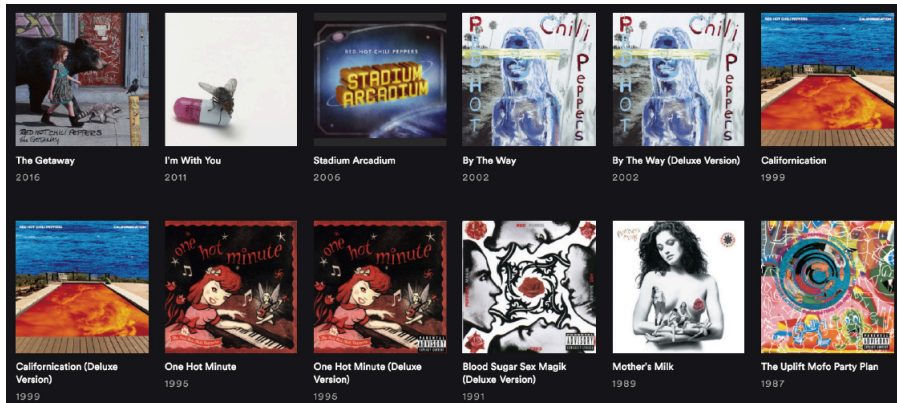
Ausgabe: Die maximale Rate, mit der Material (= Zuschauer) von der Quelle bis zur Senke (= Stadion) transportiert werden kann, ohne die Kapazitätsbeschränkungen der Straßen zu verletzen.

Andere Rechenprobleme: das CD-Brennproblem



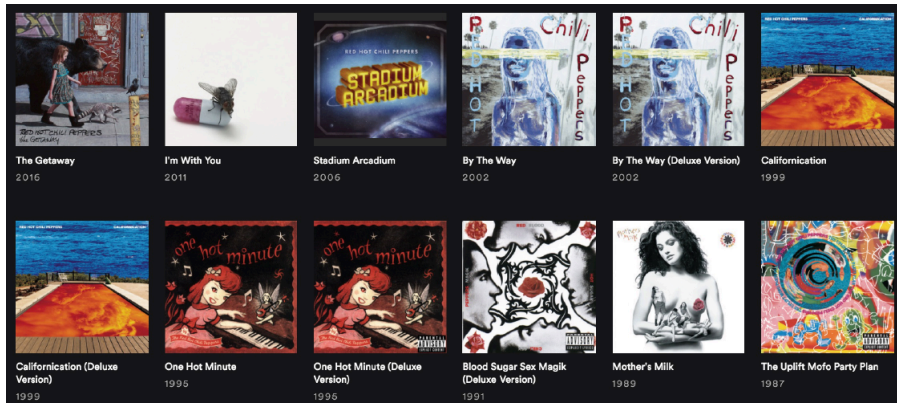
Andere Rechenprobleme: das CD-Brennproblem

Betrachte alle Schallplatten von Red Hot Chili Peppers:



Andere Rechenprobleme: das CD-Brennproblem

Betrachte alle Schallplatten von Red Hot Chili Peppers:



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 $0 < n_i \leq 80$ 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 $0 < n_i \leq 80$ Minuten,
 2. $k \in \mathbb{N}$ CDs, jeweils mit Kapazität: 80 Minuten.

- Ausgabe:** k CDs gefüllt mit einer Auswahl der N Songs, sodass
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 $0 < n_i \leq 80$ Minuten,
 2. $k \in \mathbb{N}$ CDs, jeweils mit Kapazität: 80 Minuten.

- Ausgabe:** k CDs gefüllt mit einer Auswahl der N Songs, sodass
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

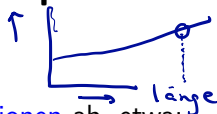
- ▶ verwendetem Computer, Programmiersprache, Fähigkeiten des Programmierers usw.

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

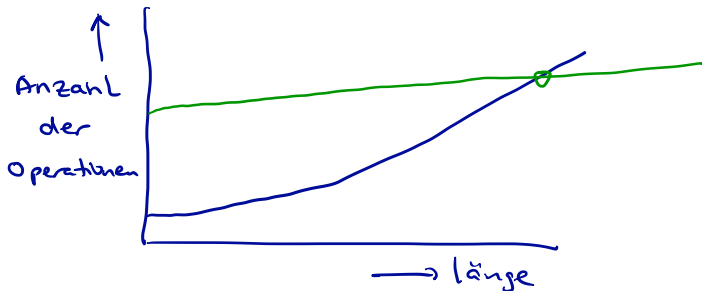
- ▶ Anzahl der elementaren Operationen sollte eine gute Abschätzung für die Anzahl der Gesamtoperationen sein.
- ▶ Anzahl der elementaren Operationen bildet die Basis zur Bestimmung der **Wachstumsrate** der Zeitkomplexität bei immer längeren Eingaben.

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	$n \cdot \log n$	•	$\in \mathbb{N}$
• $\log n$	logarithmisch	n^2	quadratisch	• $100n^2$
• n	linear	2^n	exponentiell	• $37n^2$

Zeitkomplexität in der Praxis I

Beispiel (Tatsächliche Laufzeiten)

Länge n	$33n$	$46n \log n$	Komplexität $13n^2$	$3,4n^3$	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.

$$4b \, n \, \log n$$

Länge $n_1 = 10$ $n_2 = 100$

$$\frac{\text{laufzeit für } n_2}{\text{laufzeit für } n_1} = \frac{\cancel{4b} \cdot \cancel{100} \log 100}{\cancel{4b} \cdot \cancel{10} \log 10}$$

$$= 10 \cdot \frac{\log 100}{\log 10} = 10 \cdot 2 = \textcircled{20}$$

Spalte 5 $n_1 = 10$ $n_2 = 100$

$$\frac{2^{100}}{2^{10}} = 2^{90} \quad \text{mal langsamer}$$

Zeitkomplexität in der Praxis II

Beispiel (Größte lösbare Eingabelänge)

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

$\sqrt[3]{60}$
 $\log_2 60 \approx 6$

Handwritten annotations: Green circles around '1 s', '1 m', and '1 h'. Green arrows point from the underlined values to the handwritten '60'. Green arrows point from the values 20, 26, and 32 to the handwritten '+6'.

$$13n^2$$

$$60 \cdot (\cancel{13} n_t^2) = \cancel{13} n_{\underbrace{bot}_{\uparrow min}}^2$$

$\uparrow$
sek

$$n_{bot}^2 = 60$$

$$\Rightarrow n_{bot} = \sqrt{60} \approx 7.75$$

Zeitkomplexität in der Praxis II

Beispiel (Größte lösbare Eingabelänge)

Verfügbare Zeit	Komplexität				
	$33n$	$46n \log n$	$13n^2$	$3,4n^3$	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

- 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

Idee

Wir betrachten 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 betrachten 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 betrachten 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 betrachten 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

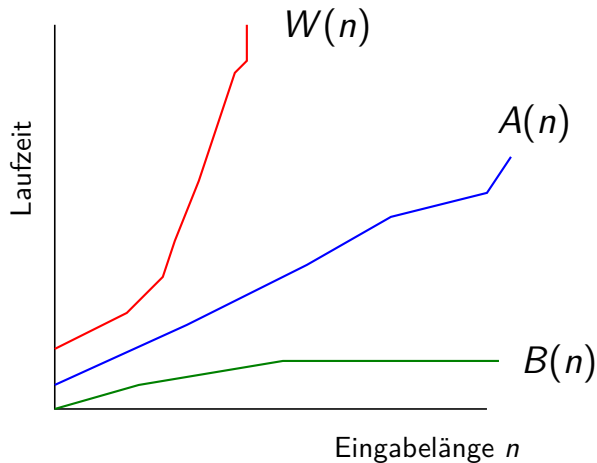
Die **Best-Case** Laufzeit von A ist die von A **minimal** benötigte Anzahl elementarer Operationen auf einer *beliebigen* Eingabe der Länge n .

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 = Menge aller Eingaben der Länge n $\in D_n$

$t(I)$ = für Eingabe I benötigte Anzahl elementarer Operationen

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

$n=3$

$010 = I \in D_3$

$\Pr(010) = \frac{1}{8}$

$\Pr(101) = \frac{1}{2}$

$\Pr(000) < 10^{-8}$

$\in D_n$

$D_3 =$

$\{ (000), (010), (100), \dots \}$

Formale Definition (I)

Einige hilfreiche Begriffe

D_n = Menge aller Eingaben der Länge n

$t(I)$ = für Eingabe I benötigte Anzahl elementarer Operationen

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

Woher kennen wir:

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

$\text{Pr}(I)$? – 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 :

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

A hand-drawn diagram in green ink. It features a vertical line on the left and a horizontal line extending to the right. At the end of the horizontal line, there is an 'x' mark. Below this 'x' mark, the text 'n=100' is written. Above the horizontal line, the equation $W(100) = \max \{ t(I) \mid I \in D_{100} \}$ is written in a similar green ink.

$$W(100) = \max \{ t(I) \mid I \in D_{100} \}$$

$n=100$

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 :

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

$$n=3 \\ \text{Pr}(010) = \frac{1}{8}$$

$$\text{Pr}(101) = \frac{1}{2}$$

$$\text{Pr}(000) = 10^{-9}$$

$$t(010) = 4$$

$$t(101) = 40$$

$$t(000) = 1$$

$$\underbrace{\frac{1}{8} \cdot 4}_{010} + \underbrace{\frac{1}{2} \cdot 40}_{101} + \underbrace{10^{-9} \cdot 1}_{000} + \dots$$

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?

```
1 bool linSearch(int E[], int n, int K) {  
2     for (int index = 0; index < n; index++) {  
3         if (E[index] == K) {  
4             return true; // oder: return index;  
5         }  
6     }  
7     return false; // nicht gefunden  
8 }
```

Lineare Suche – Analyse

Elementare Operation

Vergleich einer ganzen Zahl K mit Element $E[\text{index}]$.

Lineare Suche – Analyse

Elementare Operation

Vergleich einer ganzen Zahl K mit Element $E[\text{index}]$.

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 $E[\text{index}]$.

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 $E[\text{index}]$.

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 $K == E[n-1]$).

Lineare Suche – Analyse

Elementare Operation

Vergleich einer ganzen Zahl K mit Element $E[\text{index}]$.

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 $K == E[n-1]$).
- ▶ $B(n) = 1$, da ein Vergleich ausreicht, wenn K gleich $E[0]$ ist.

Lineare Suche – Analyse

Elementare Operation

Vergleich einer ganzen Zahl K mit Element $E[\text{index}]$.

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 $K == E[n-1]$).
- ▶ $B(n) = 1$, da ein Vergleich ausreicht, wenn K gleich $E[0]$ ist.
- ▶ $A(n) \approx \frac{1}{2}n?$, da im Schnitt K mit etwa der Hälfte der Elemente im 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 ".

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

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:

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

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:

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

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:

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

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:

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

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:

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

Herleitung

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

Herleitung

$$\begin{aligned} A(n) &= \Pr\{K \text{ in } E\} \cdot A_{K \in E}(n) + \Pr\{K \text{ nicht in } E\} \cdot A_{K \notin E}(n) \\ &\quad \left| \quad A_{K \in E}(n) = \frac{n+1}{2} \right. \\ &= \Pr\{K \text{ in } E\} \cdot \frac{n+1}{2} + \Pr\{K \text{ nicht in } E\} \cdot A_{K \notin E}(n) \end{aligned}$$

Herleitung

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

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

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

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

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

Herleitung

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

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

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

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

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

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

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

Herleitung

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

$$\quad \mid \quad A_{K \in E}(n) = \frac{n+1}{2}$$

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

$$\quad \mid \quad \Pr\{\text{nicht } B\} = 1 - \Pr\{B\}$$

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

$$\quad \mid \quad A_{K \notin E}(n) = n$$

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

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

Lineare Suche – Average-Case-Analyse

Endergebnis

Die Average-Case-Zeitkomplexität der linearen Suche ist:

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

Lineare Suche – Average-Case-Analyse

Endergebnis

Die Average-Case-Zeitkomplexität der linearen Suche ist:

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

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:

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

Beispiel

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

= 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:

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

Beispiel

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

- = 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

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 (Teil I)

1. Algorithmische Komplexität
2. Asymptotische Effizienz
3. Elementare Datenstrukturen
4. Suchen
5. Rekursionsgleichungen
6. Sortieren: in-situ, Mergesort, Heapsort, Quicksort

Übersicht (Teil II)

1. Binäre Suchbäume
2. Rot-Schwarz-Bäume
3. Hashing
4. Elementare Graphenalgorithmen
5. Minimale Spannbäume
6. Kürzeste-Pfade-Algorithmen
7. Maximale Flüsse
8. Dynamische Programmierung
9. 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 , 4. Auflage.



Wichtige Termine

Vorlesungstermine

Vorlesung: Mo. 08:30–10:00, H01

Fr. 13:15–14:45, H01

Keine Vorlesung am 16.04., 04.05., 13.07., 16.07., 20.07.

Letzte Vorlesung am 9. Juli 2018

Wichtige Termine

Vorlesungstermine

Vorlesung: Mo. 08:30–10:00, H01

Fr. 13:15–14:45, H01

Keine Vorlesung am 16.04., 04.05., 13.07., 16.07., 20.07.

Letzte Vorlesung am 9. Juli 2018

Frontalübung: Di. 14:15–15:45, Aula (Hauptgebäude)

Am 04.05. findet zur Vorlesungszeit eine Frontalübung statt

Erste Frontalübung: Di. 24. April

Übungsbetrieb

Übungsgruppen

- ▶ 24 Übungsgruppen: verschiedene Uhrzeiten Mo.–Fr.
- ▶ Spezialübung für Lehramtsstudenten und CES Studierende
- ▶ Einige Übungsgruppen für Erstsemester
- ▶ Koordinatoren: [Sebastian Junges](#), [Benjamin Kaminski](#), [David Korzeniewski](#) und [Tim Quatmann](#).

Übungsbetrieb

Übungsgruppen

- ▶ 24 Übungsgruppen: verschiedene Uhrzeiten Mo.–Fr.
- ▶ Spezialübung für Lehramtsstudenten und CES Studierende
- ▶ Einige Übungsgruppen für Erstsemester
- ▶ Koordinatoren: [Sebastian Junges](#), [Benjamin Kaminski](#), [David Korzeniewski](#) und [Tim Quatmann](#).

Anmeldung für die Übungsgruppen

Anmeldung zum Übungsbetrieb über Web-link

<https://lufgi2.informatik.rwth-aachen.de/dsal18/>

bis **Mittwoch, 18. April 2018, 12:00 Uhr (Aachener Zeit)**.

Übungsbetrieb

Wichtige Termine

Anmeldung: 18. April 2018

1.Übungszettel: 13. April 2018 **heute**

Abgabe 1. Übungszettel: Donnerstag, 19. April 2018

2.Übungszettel: 19. April 2018

Abgabe 2. Übungszettel: Donnerstag, 26. April 2018, etc.

3.Übungszettel: 26. April 2018

.....

Frontalübung: Dienstag, 14:15–15:45 **ab 24. April 2015**

Übungsgruppen: Montag–Freitag, (fast) jederzeit **ab 23. April**

Präsenzübung: Donnerstag, 07. Juni 2018 (abends?)

Prüfung

Die Prüfung ist eine schriftliche Klausur von 120 Minuten.

Zulassungskriterium Klausur

1. Mindestens 50% aller in den Übungen erreichbaren Punkte **bis** PÜ
2. Mindestens 50% aller in den Übungen erreichbaren Punkte **ab** PÜ
3. Mindestens 50% der in der Präsenzübung (PÜ) erreichbaren Punkte.

CES-Studenten brauchen **kein** Zulassungskriterium zu erfüllen.

Bonusregelungen

1. Bei $\geq 70\%$ in **allen drei Punkten**: eine Notenstufe besser (außer bei 1.0 und 5.0).
2. 5 Bonuspunkte pro Übungsblatt in **LaTeX**; also $n+5$ (von 100) Punkte statt n Punkte.

Wichtige Termine

Wichtige Termine

Präsenzübung: Donnerstag, 7. Juni 2018 (abends?)

Klausur: Montag, 13. August 2018 (11:00-14:00)

Wiederholungsklausur: Montag, 10. September 2018 (10:00-13:00)

Wichtige Termine

Wichtige Termine

Präsenzübung: Donnerstag, 7. Juni 2018 (abends?)

Klausur: Montag, 13. August 2018 (11:00-14:00)

Wiederholungsklausur: Montag, 10. September 2018 (10:00-13:00)

Anmeldung zur Prüfung über CAMPUS-Office bis **18.05., 23:59 Uhr.**

Sonstiges

Mehr Information

- ▶ Webseite:
`https://moves.rwth-aachen.de/teaching/ss-18/dsal/`
- ▶ E-Mail: `dsal18@i2.informatik.rwth-aachen.de`

Nächste Vorlesung

Freitag 20. April, 13:15.