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-20/datenstrukturen-und-algorithmen/

06. April 2020

RWTH AACHEN UNIVERSITY

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

update in Vergleich zu der Vorlesungsvideo

Übersicht

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 $\langle a_1, a_2, \ldots, a_n \rangle$ mit $a_i \in \mathbb{N}$.

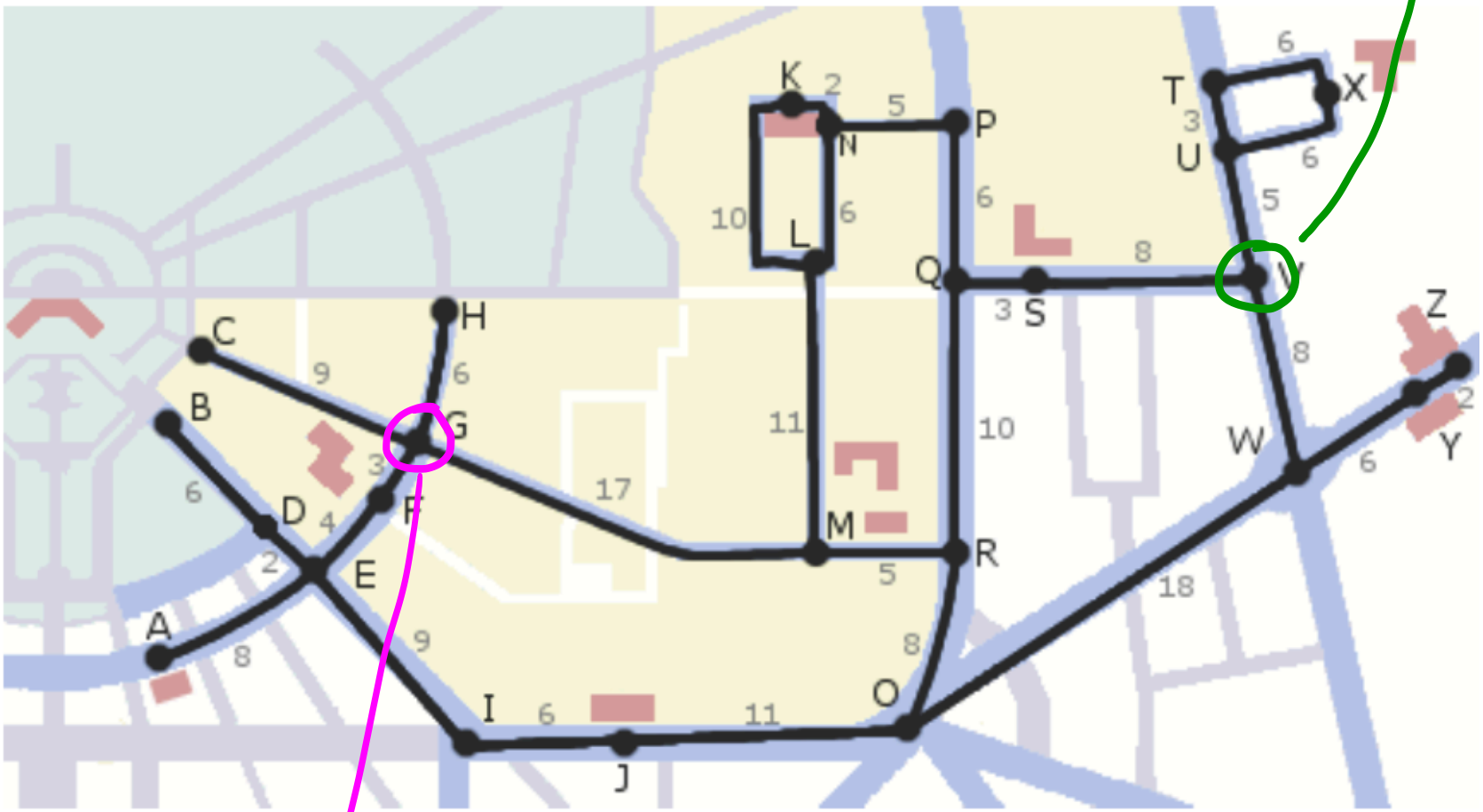
Ausgabe: Eine Permutation (Umordnung) $\langle b_1, b_2, \ldots, b_n \rangle$ der Eingabefolge, sodass $b_1 \leqslant b_2 \leqslant \ldots \leqslant b_n$.

$\langle a_1, a_2, a_3, a_4 \rangle = \langle 3, 21, 1, 18 \rangle$

$\langle b_1, \ldots, b_4 \rangle = \langle 1, 3, 18, 21 \rangle$

Andere Rechenprobleme: kürzester Weg

Andere Rechenprobleme: kürzester Weg



Andere Rechenprobleme: kürzester Weg

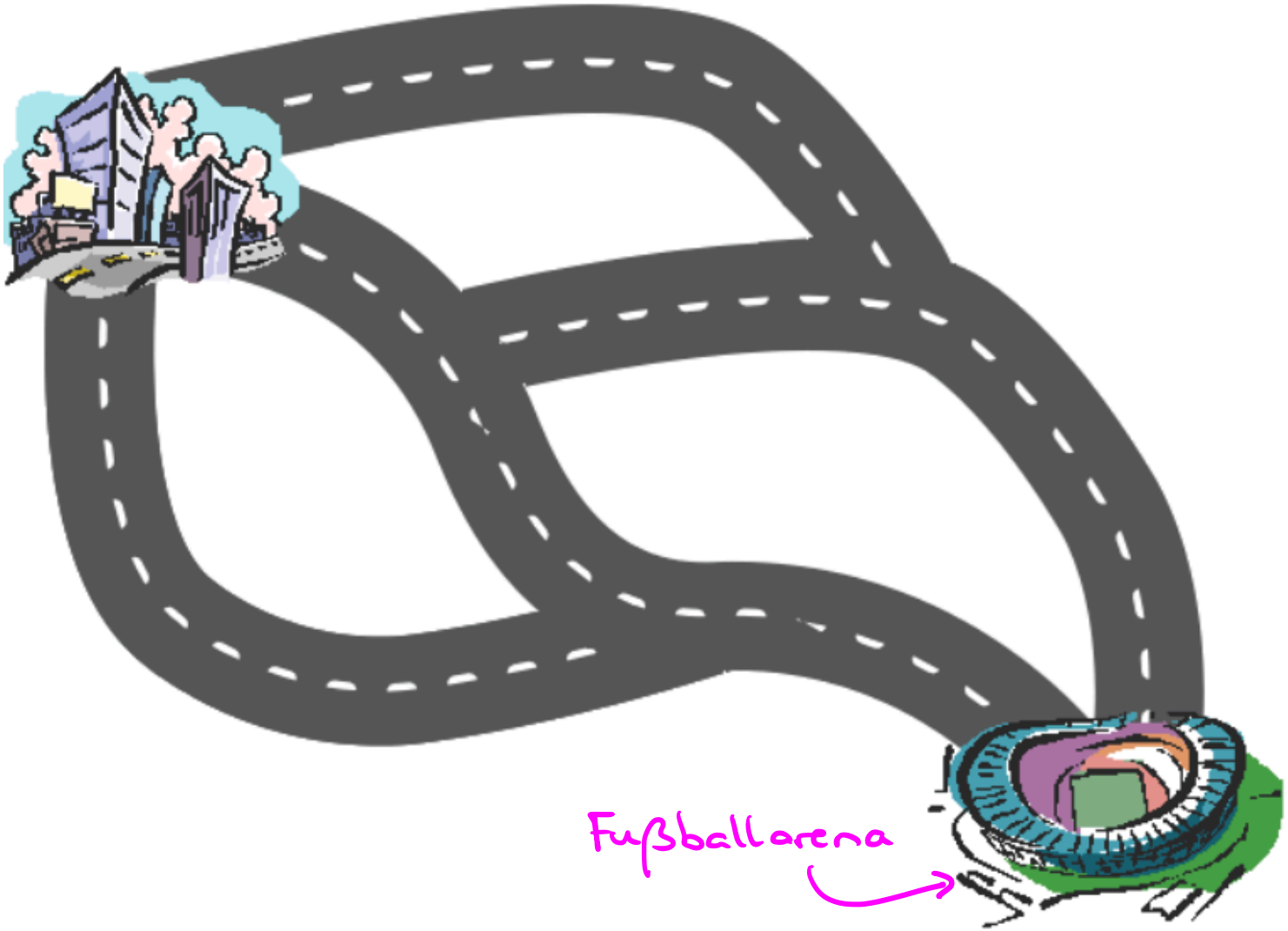
Beispiel (kürzester Weg)

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

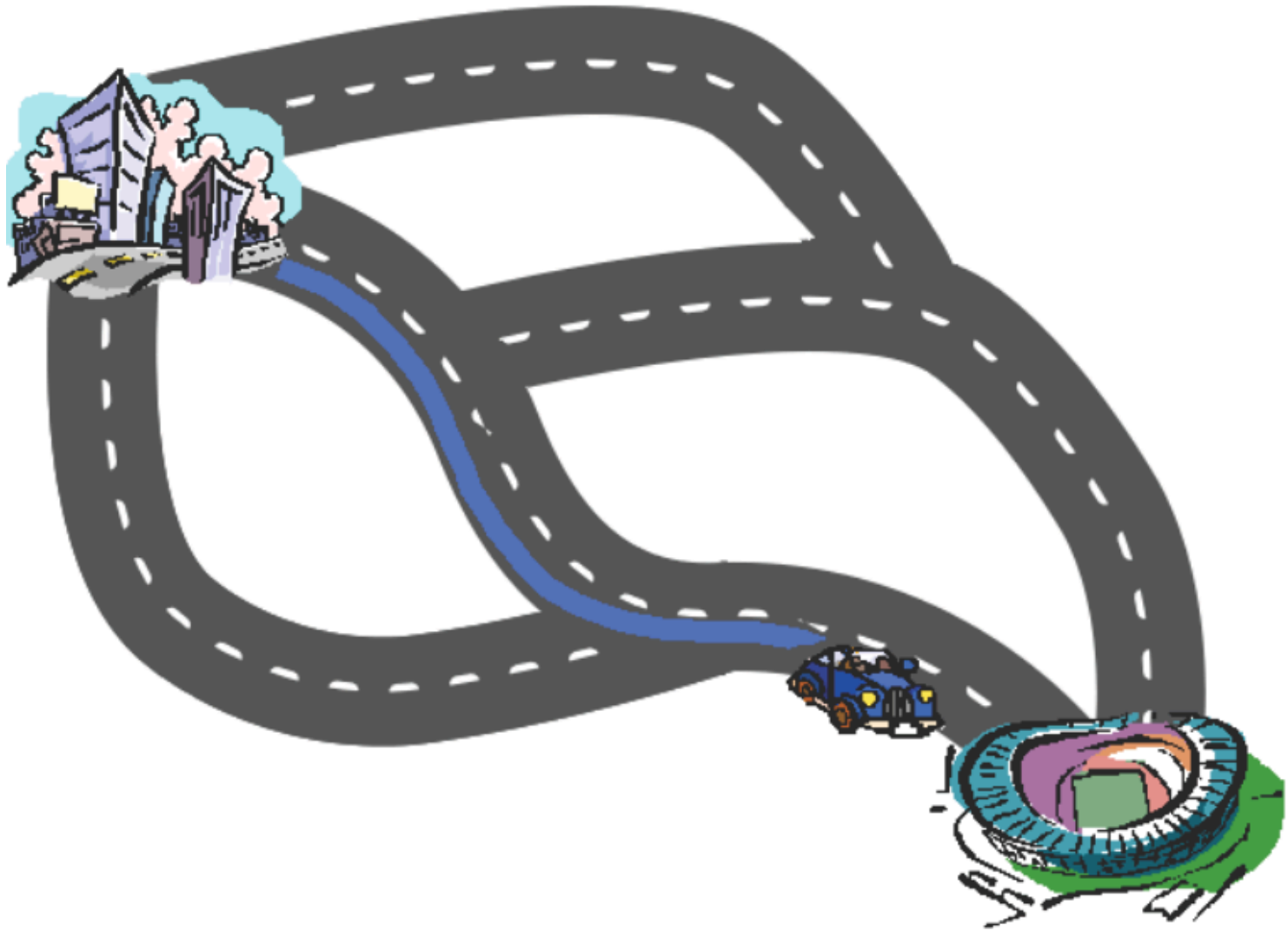
Ausgabe: Ein kürzeste Weg von s nach z.

es kann mehrere kürzeste Wege geben

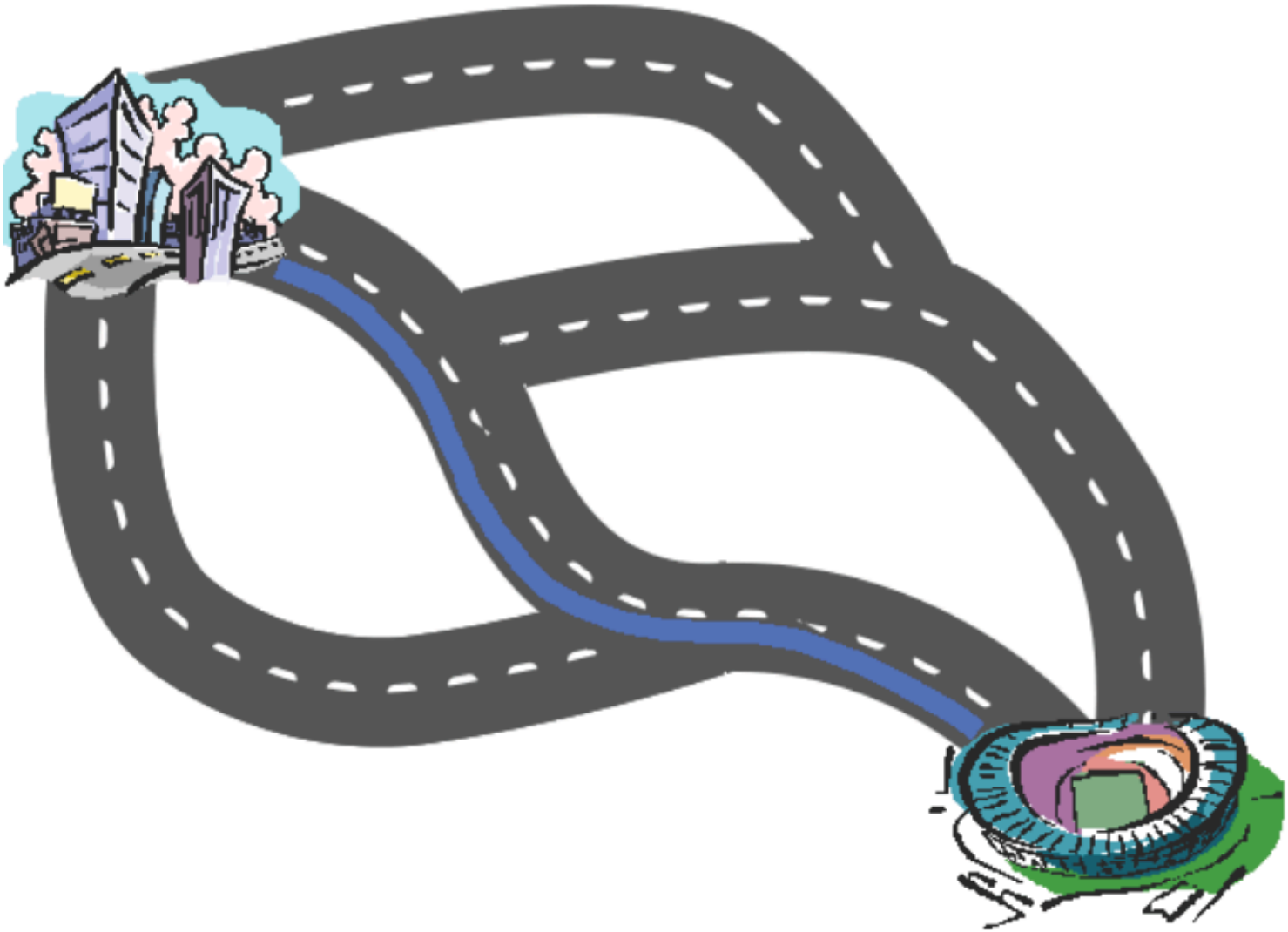
Andere Rechenprobleme: maximale Flüsse



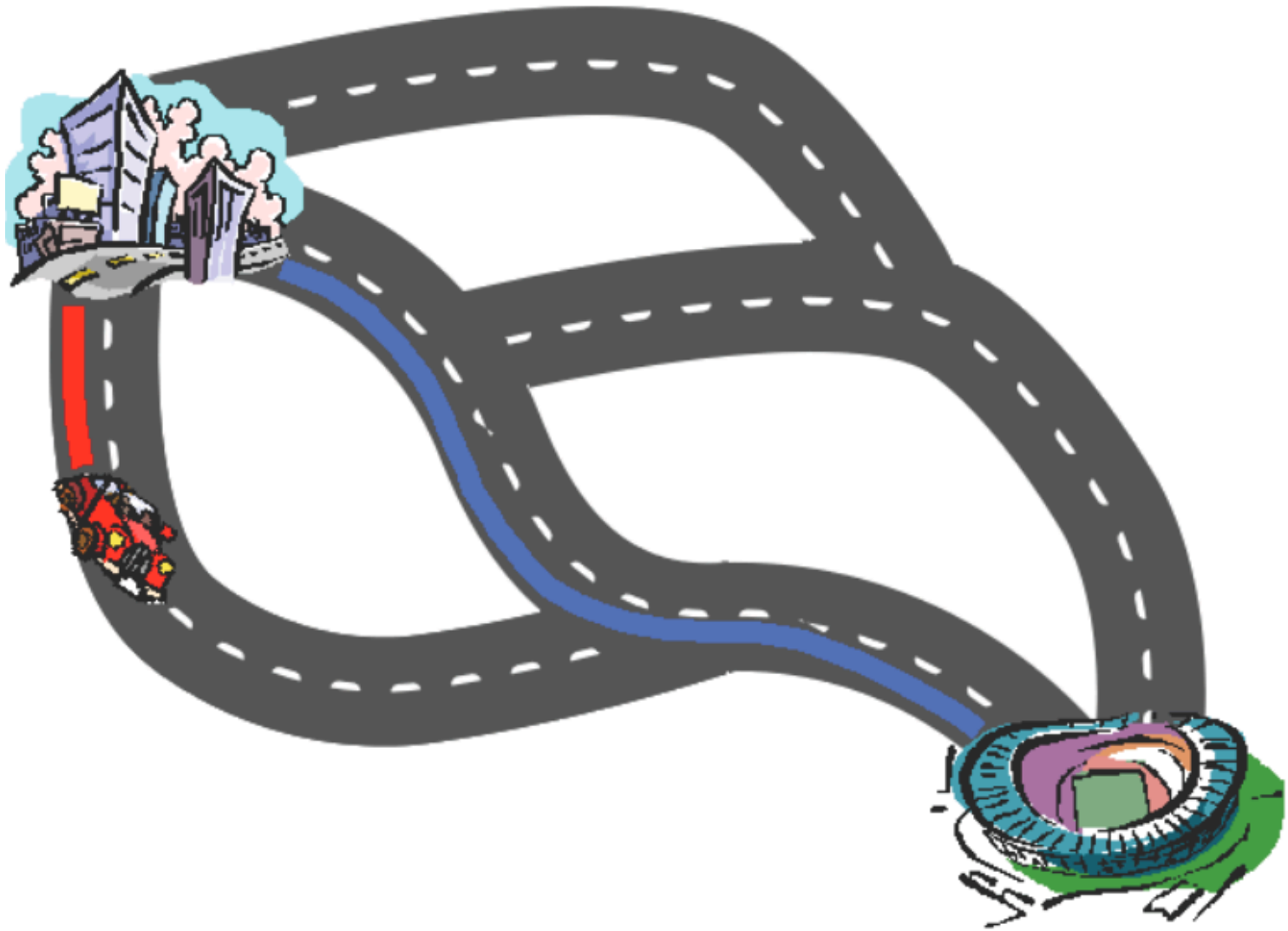
Andere Rechenprobleme: maximale Flüsse



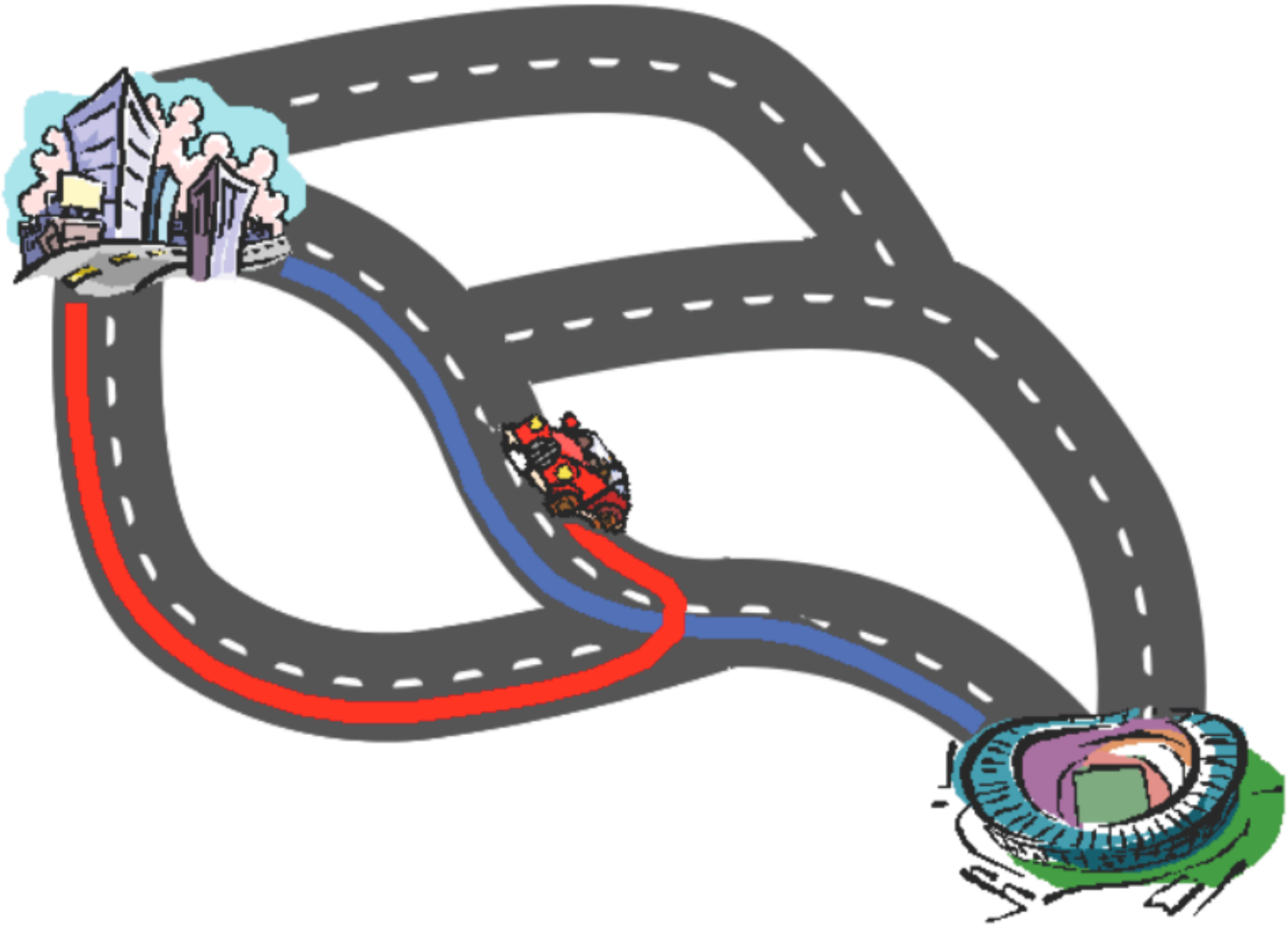
Andere Rechenprobleme: maximale Flüsse



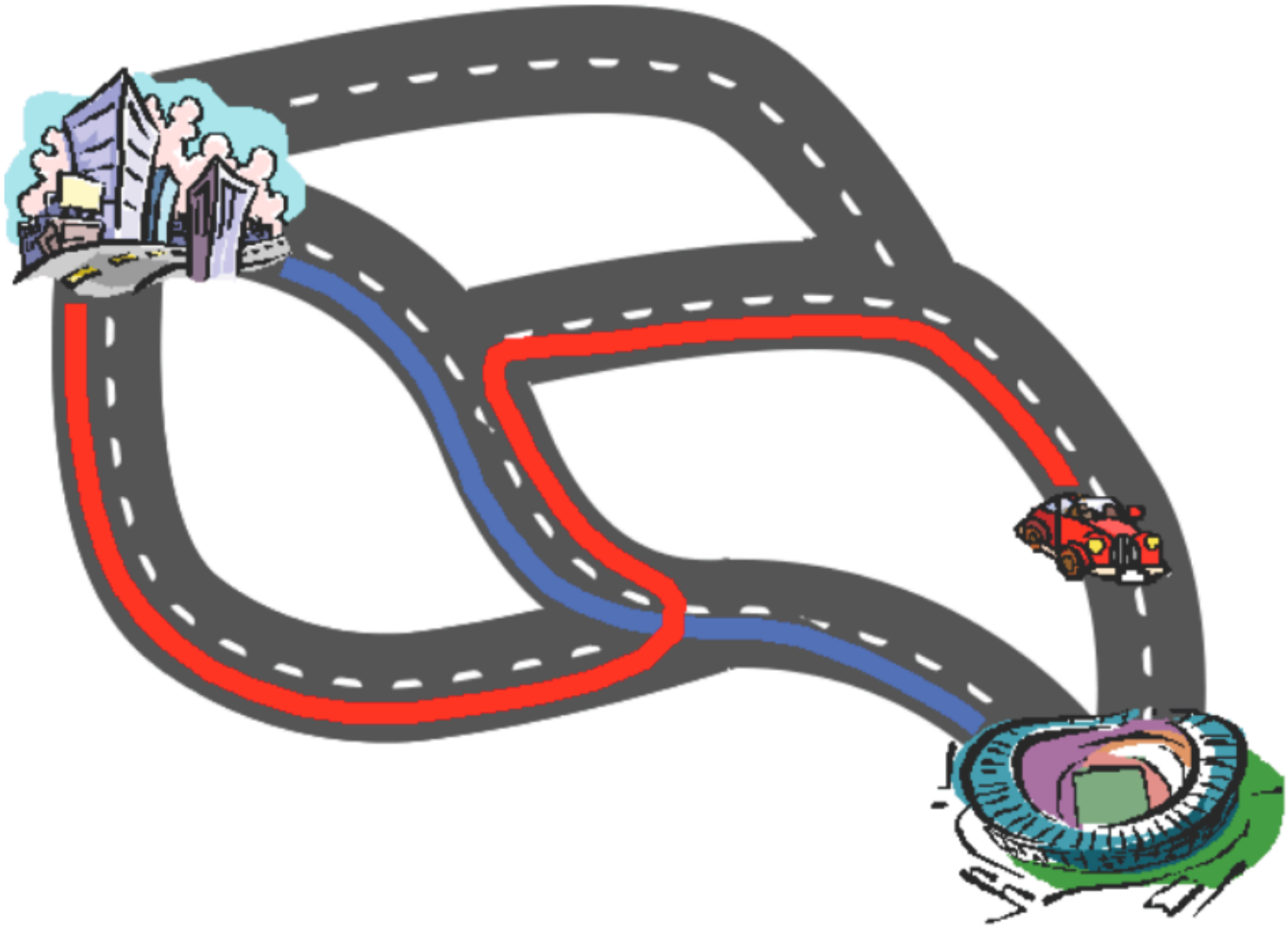
Andere Rechenprobleme: maximale Flüsse



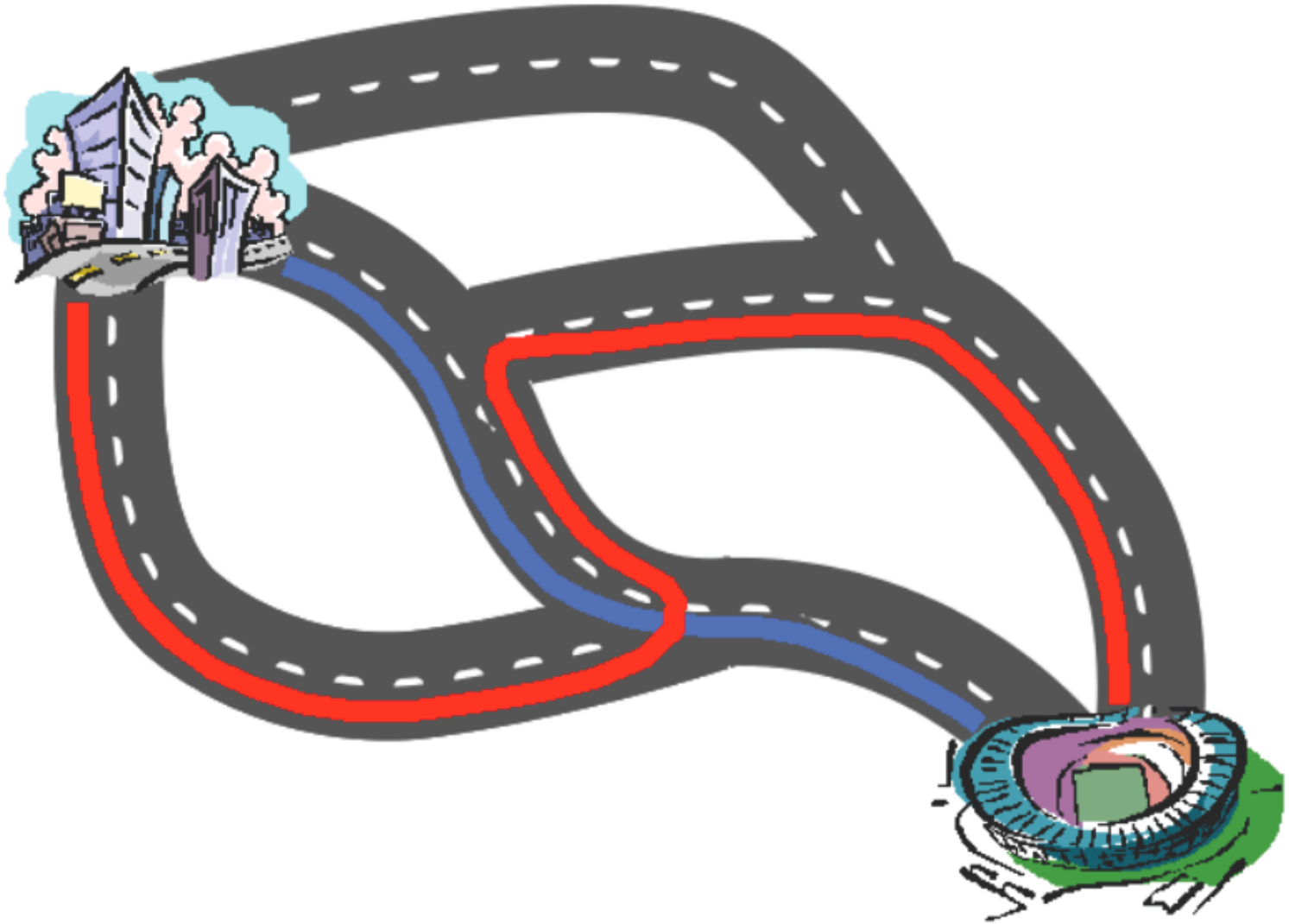
Andere Rechenprobleme: maximale Flüsse



Andere Rechenprobleme: maximale Flüsse



Andere Rechenprobleme: maximale Flüsse



Andere Rechenprobleme: maximale Flüsse

Beispiel (maximale Flüsse)

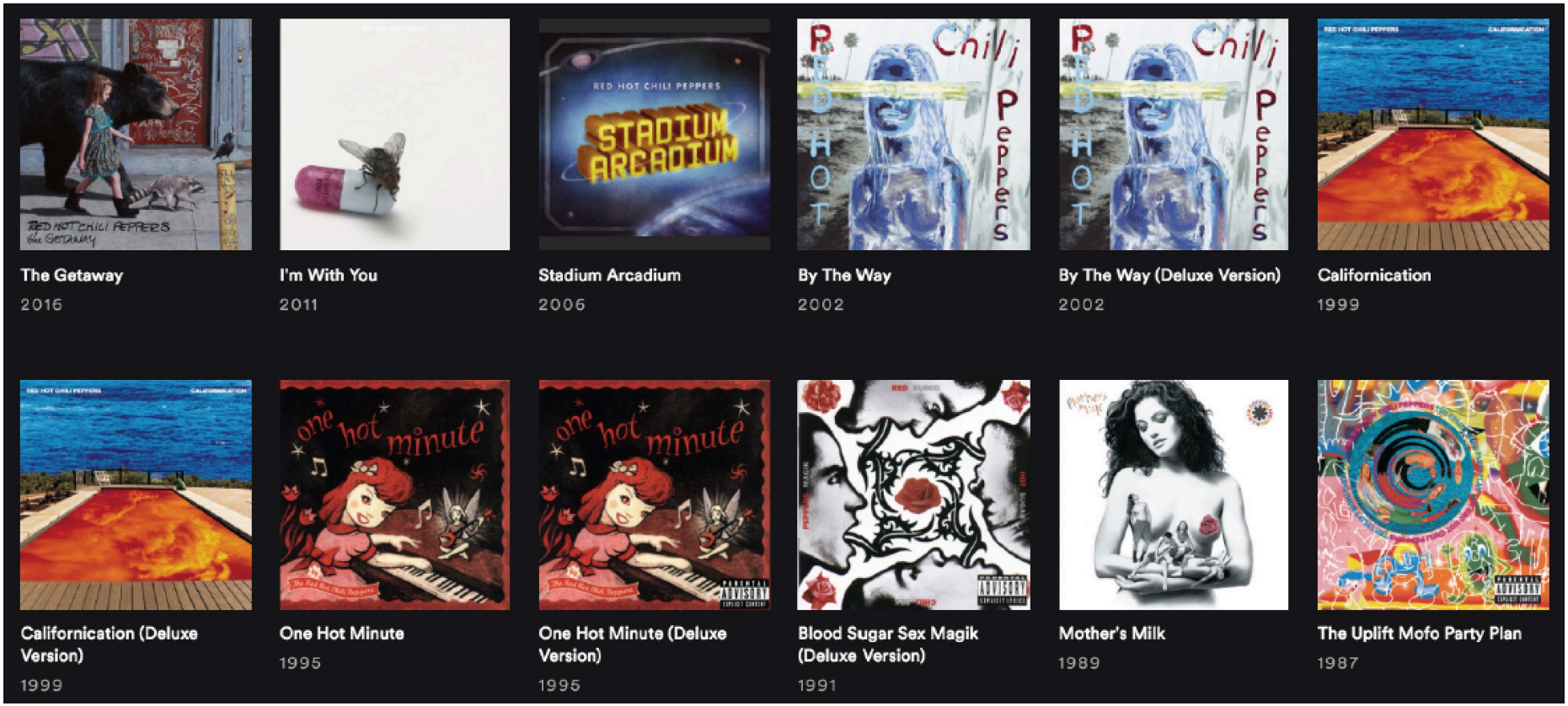
Eingabe:
1. Eine Straßenkarte, auf der die Kapazität der Straßen eingezeichnet ist,
2. eine Quelle und
3. eine Senke.

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:



Wie bekommen wir eine Kompilation ihrer Songs auf einige CDs?

(aus den Zeiten dass es keine Streaming gab...)

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, 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.

Betrachte folgende Strategie:

① Sortiere die N Songs nach Dauer

② Wähle die m kürzeste Songs, sodaß die gerade noch auf k CDs passen:

$$\sum_{i=1}^{m} n_i \leq k \cdot 80 \quad \text{und} \quad \sum_{i=1}^{m+1} n_i > k \cdot 80$$

③ Sortiere diese m Songs chronologisch

④ Brenne die k CDs, so lange es geht.

Frage: ist diese Strategie optimal?

Algorithmen

z.B. kurz, gut verständlich

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

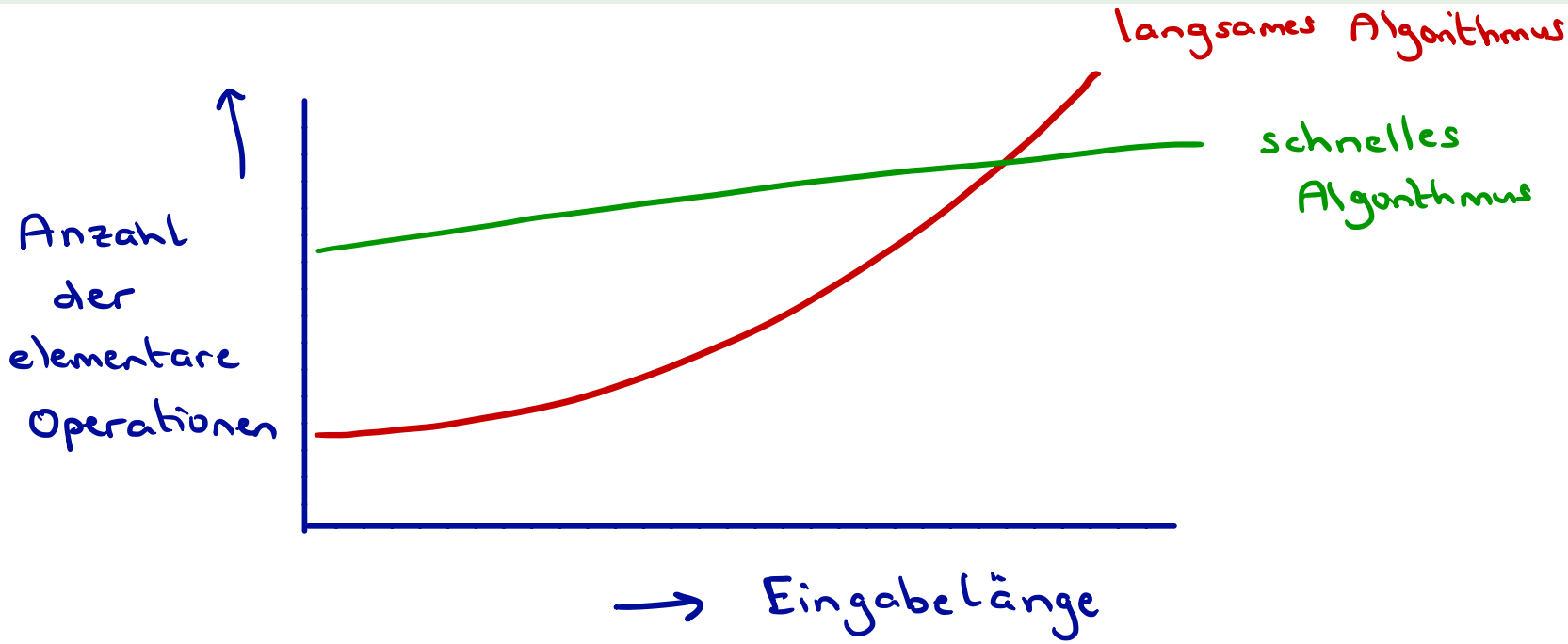
- Anzahl der elementaren Operationen sollte eine gute Abschätzung für die Anzahl der Gesamtoperationen sein.
- Anzahl der elementaren Operationen bildet die Basis zur Bestimmung der Wachstumsrate der Zeitkomplexität bei immer längeren Eingaben.

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 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.

2. Spalte: $46\ n \log n$

Sei $n_1 = 10$ und $n_2 = 100$

Dann:

$$\frac{\text{Laufzeit für } n_2}{\text{Laufzeit für } n_1} = \frac{46 \cdot (100 \log 100)}{46\ (10 \log 10)}$$

$$= 10\ \frac{\log 100}{\log 10} = 20$$

(mal langsamer)

$n_3 = 1000$

$$\frac{\text{Laufzeit für } n_3}{\text{Laufzeit für } n_2} = 10\ \frac{\log 1000}{\log 100} = 15$$

usw.

5. Spalte: 2^n

$$\frac{\text{Laufzeit für } n_2}{\text{Laufzeit für } n_1} = \frac{2^{100}}{2^{10}} = 2^{90}$$

n_3: $\quad \frac{2^{1000}}{2^{100}} = 2^{900}$

(mal langsamer)

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				

Sei t die verfügbare Zeit

n_t = maximale Eingabegröße in t Zeiteinheiten lösbar.

F: wie groß ist n_{bot}, d.h. die maximale Eingabegröße wenn $bo \cdot t$ Zeit zur Verfügung steht?

3. Spalte $46\, n \log n$ $t = 1$ sekunde

$$bo \cdot (46\, n_t \log n_t) = 46 \cdot n_{bo \cdot t} \log n_{bot}$$

$$\Leftrightarrow \quad bo \cdot (2000 \log 2000) = n_{bo \cdot t} \log n_{bo \cdot t}$$

$$\Leftrightarrow \quad n_{bo \cdot t} \approx 82.000$$

4. Spalte n^3

$$bo \cdot 3{,}4\, n_t^3 = 3{,}4\, n_{o \cdot t}^3$$

$$\Leftrightarrow \quad n_{bo \cdot t} = \sqrt[3]{bo} \cdot n$$

$$\Leftrightarrow \quad n_{bo \cdot t} \approx 3{,}88 \cdot t$$

5. Spalte $bo \cdot 2^{n_t} = 2^{n_{bot}}$

$$\Leftrightarrow 2^{\log_2 bo} \cdot 2^{n_t} = 2^{n_{bo \cdot t}} \Leftrightarrow n_{bo \cdot t} = n_t + \log_2 bo$$

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	$N + \log K$

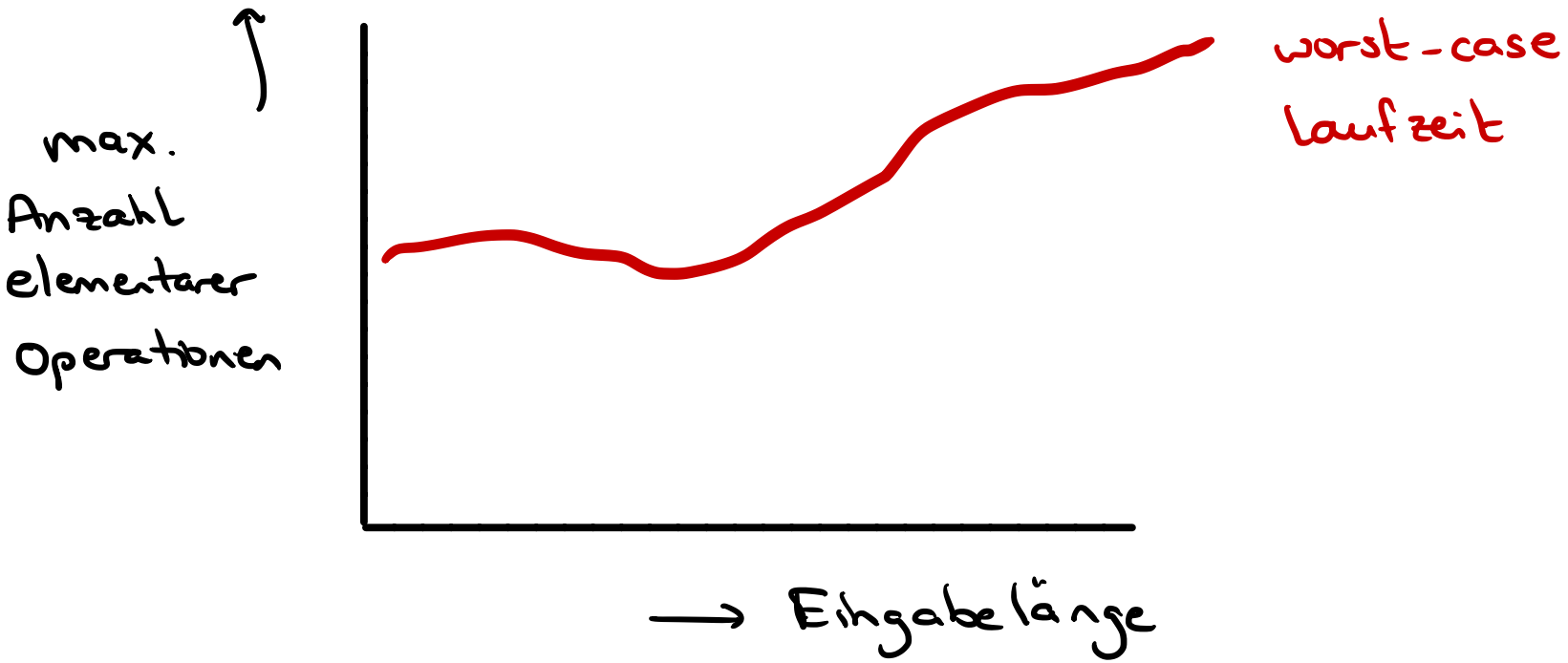
Übersicht

Idee

Wir 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

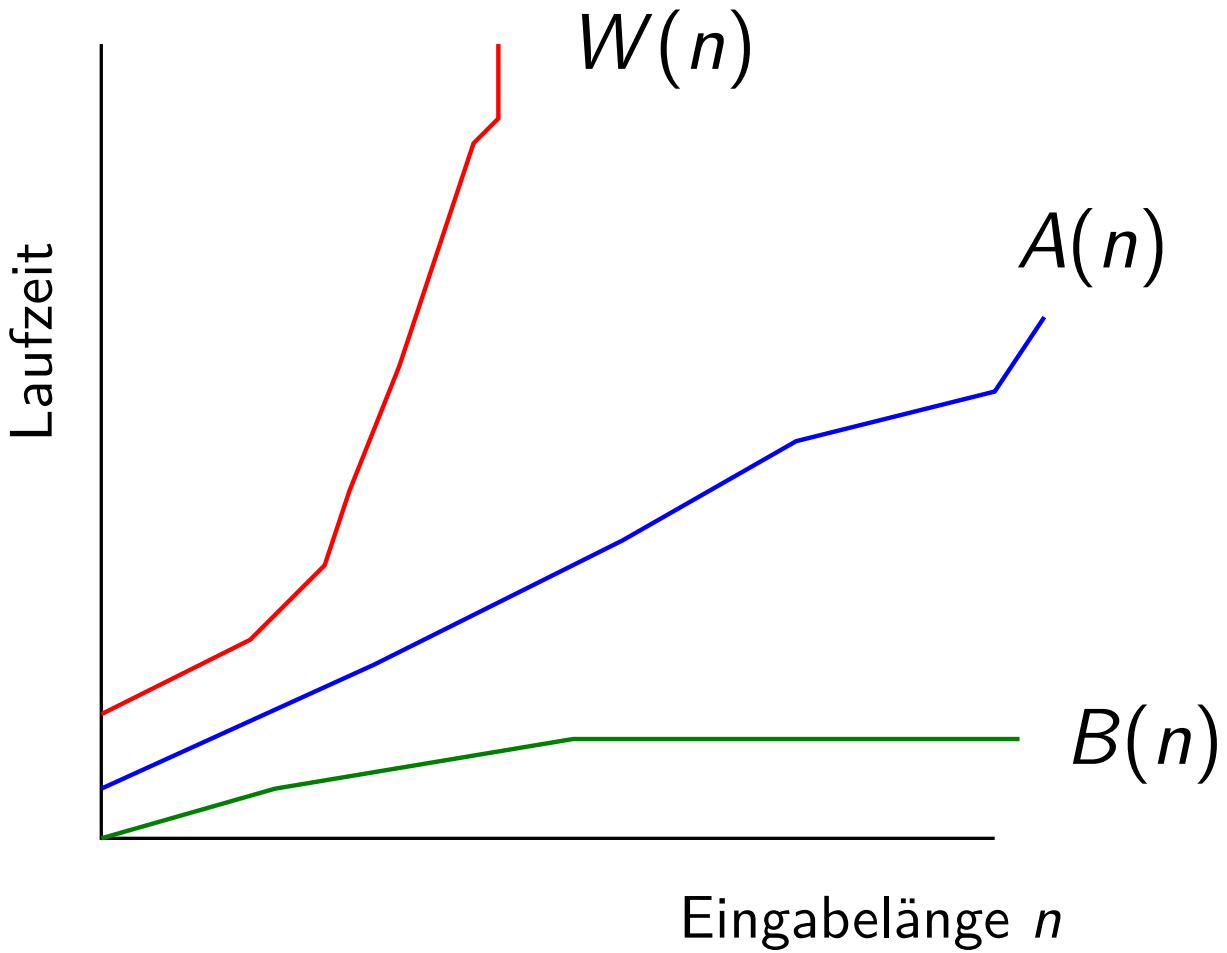
Die Best-Case Laufzeit von A ist die von A minimal benötigte Anzahl elementarer Operationen auf einer *beliebigen* Eingabe der Länge n.

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

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

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

nur benötigt für average-case Laufzeit

Formale Definition (I)

"zähle" Anzahl elementarer Operationen im Algorithmus

Einige hilfreiche Begriffe

D_n = Menge aller Eingaben der Länge n

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

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

Woher kennen wir:

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

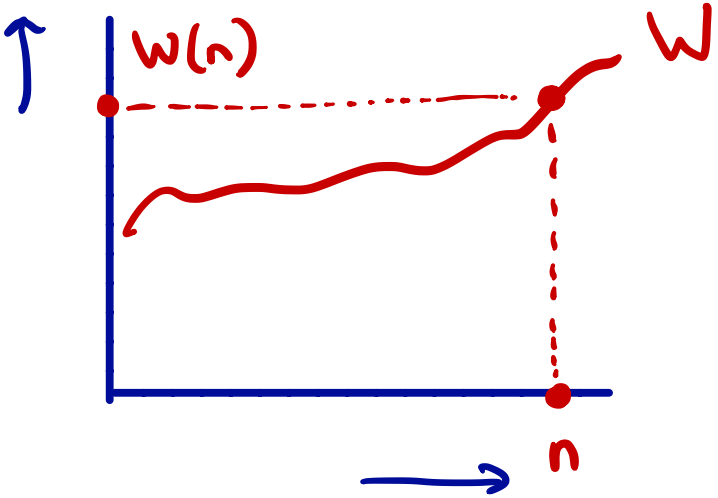
$\Pr(I)$? – Erfahrung, Vermutung (z. B. „alle Eingaben treten mit gleicher Wahrscheinlichkeit auf“).

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:

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



benötigte Anzahl elementarer Operationen für alle mögliche Eingaben der Länge 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:

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

z.B. sei $n=3$. Eingaben: Bitvektoren der Länge n.

$\Pr(010) = \frac{1}{8}$ $\Pr(101) = \frac{1}{2}$ $\Pr(000) = 10^{-9}$

$t(010) = 4$ $t(101) = 1$ $t(000) = 4000$

$A(3) = \frac{1}{8} \cdot 4 + \frac{1}{2} \cdot 1 + 10^{-9} \cdot 4000 + \ldots$

Lineare Suche : Beispiel für W(n), B(n) und A(n)

Rechenproblem

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

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

Algorithmus:

```
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

E[index] == K

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-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.

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.
- 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: $\sum_{I \in D_n} Pr(I \mid k \in E) \cdot t(I)$

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

Herleitung

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

$$\left| \; A_{K\in E}(n) = \frac{n+1}{2} \right. \text{, s. Folie 41}$$

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

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

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

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

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

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

Lineare Suche – Average-Case-Analyse

Endergebnis

Die Average-Case-Zeitkomplexität der linearen Suche ist:

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

Beispiel

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

- $= 1$, dann $A(n) = \frac{n+1}{2}$, d. h. etwa 50% von $\mathtt{E}$ ist überprüft.
- $= 0$, dann $A(n) = n = W(n)$, d. h. $\mathtt{E}$ wird komplett überprüft.
- $= \frac{1}{2}$, dann $A(n) = \frac{3 \cdot n}{4} + \frac{1}{4}$, d. h. etwa 75% von $\mathtt{E}$ 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

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 (2013).

Video-Vorlesungen

- Die Vorlesungen finden im SoSe 2020 voraussichtlich nur in digitaler Form statt
- Also bis auf weiteres: keine Vorlesungen mit Präsenz
- Die Vorlesungen sind als Video (Aufnahmen VideoAG 2015) verfügbar:

  `https://video.fsmpi.rwth-aachen.de/15ss-dsal`

Folien

- Für jede Vorlesung gibt es aktuelle Vorlesungsfolien
  - die können ggf. leicht abweichen von den Video-Folien
  - es gilt: der Inhalt der aktuellen Folien ist gültig

- Die Folien werden an den Vorlesungstagen (Mo.+Fr.) bereit gestellt
  wobei Feiertage (z.B. Ostermontag, 1. Mai) ausfallen

- 21 Vorlesungstermine:

April 6.4, 17.4, 20.4, 24.4, 27.4,
Mai 4.5, 8.5, 11.5, 15.5, 18.5, 22.5, 25.5,
Juni 8.6, 12.6, 15.6, 19.6, 22.6, 26.6, 29.6,
Juli 3.7, 6.7

Frontalübung

Frontalübung: Mi. 12:30–14:00, über Zoom

Erste Frontalübung: Mi. 15. April

Fragestunde: , über Zoom (ggf. im Zukunft; falls ungesetzt werden Details später bekannt gegeben)

Erste Fragestunde: NN. April

Übungsbetrieb

Übungsgruppen

- 25 Übungsgruppen: verschiedene Uhrzeiten Mo.–Fr.
- Koordinatoren: Stefan Dollase, Ira Fesefeldt, Tim Quatmann, Jip Spel und Tobias Winkler.

Anmeldung für die Übungsgruppen

Anmeldung zum Übungsbetrieb über RWTHOnline bis **Mittwoch, 8. April 2020, 23:55 Uhr (Aachener Zeit).**

Übungsbetrieb (1)

Es gibt Übungsblätter, die wöchentlich ausgegeben werden.

- Ausgabe spätestens Montags über RWTHmoodle
- Bearbeitung (und Abgabe) in Gruppen von je drei Studierende aus dem selben Tutorium[1]
- Abgabe der Hausaufgaben:
  - Frist: Montags um 08:00 Uhr
  - Format: elektronisch als pdf-Datei über RWTHmoodle
- Hausaufgaben werden korrigiert, sind aber nicht für Klausurzulassung gedacht
- Jedes Übungsblatt hat $\geqslant 1$ klar identifizierte "Klausur"-Aufgaben
  - Aufgaben die vom Niveau her einer Klausuraufgabe entsprechen
- Bearbeitung der Übungsblätter wird dringend empfohlen!
  - Tipp: ohne Übung, keine Chance die Klausur zu bestehen

[1] Suche nach Abgabepartner/inn/en über das entsprechende Forum im Moodle Lernraum.

Übungsbetrieb (2)

- Übungsblätter oben mit Übungsgruppe, Name und Matrikelnummer beschriften
- Musterlösungen werden mittels Video-Aufzeichnungen vorgerechnet
- Musterlösungen werden als pdf-Datei zur Verfügung gestellt
- In der Frontalübung wird:
  - extra Stoff vorgestellt und
  - zusätzliche Aufgaben vorgerechnet

Übungsbetrieb (3)

Wichtige Termine

Anmeldung:	8. April 2020
1.Übungszettel:	6. April 2020 heute
Abgabe 1. Übungszettel:	Dienstag, 14. April 2020 (08:00 Uhr)
2.Übungszettel:	14. April 2020
Abgabe 2. Übungszettel:	Montag, 20. April 2020 (08:00 Uhr), etc.
3.Übungszettel:	21. April 2020
	.....
Frontalübung:	Mittwoch, 12:30–14:00 ab 15. April 2020
Präsenzübung:	Freitag, 29. Mai 2020 (nachmittags)

Prüfung

Die Prüfung ist (voraussichtlich) eine schriftliche Klausur von 120 Minuten.

Zulassungskriterium Klausur

Mindestens 50% der in der Präsenzübung (PÜ) erreichbaren Punkte.

Wichtige Termine

Präsenzübung:	Freitag, 29. Mai 2020
Klausur:	Freitag, 14. August 2020
Wiederholungsklausur:	Mittwoch, 09. September 2020

Anmeldung zur Prüfung über RWTH-Online **01.05., 23:55 Uhr**.

Anmeldungen

Zusammenfassend die erforderlichen Anmeldungen:

Vorlesung: Keine Anmeldung erforderlich.

Übungsgruppen:

Anmeldung zum Übungsbetrieb über RWTHOnline
Frist: bis Mi., 8. April 2020 um 23:55 Uhr.

Präsenzübung:

Anmeldung zur Präsenzübung über RWTHMoodle
Frist: zwischen 27. April und 24. Mai um 23:55 Uhr

Klausur:

Anmeldung zur Prüfung über RWTHOnline
Frist: ab Fr., 01. Mai

Fragen

1. Erster Ansprechpartner ist der Tutor Ihrer Übungsgruppe

3. Diskussionsforum über RWTHmoodle

4. E-Mail: `dsal20@i2.informatik.rwth-aachen.de`

5. Bitte keine Email direkt an mich oder die Assistenten

6. Zur Not einen Zoom-Termin mit mir vereinbaren.

Nächste Vorlesung

Nächste Vorlesung

Freitag 17. April.