



Technische
Universität
Braunschweig



Algorithmen und Datenstrukturen – Übung #2

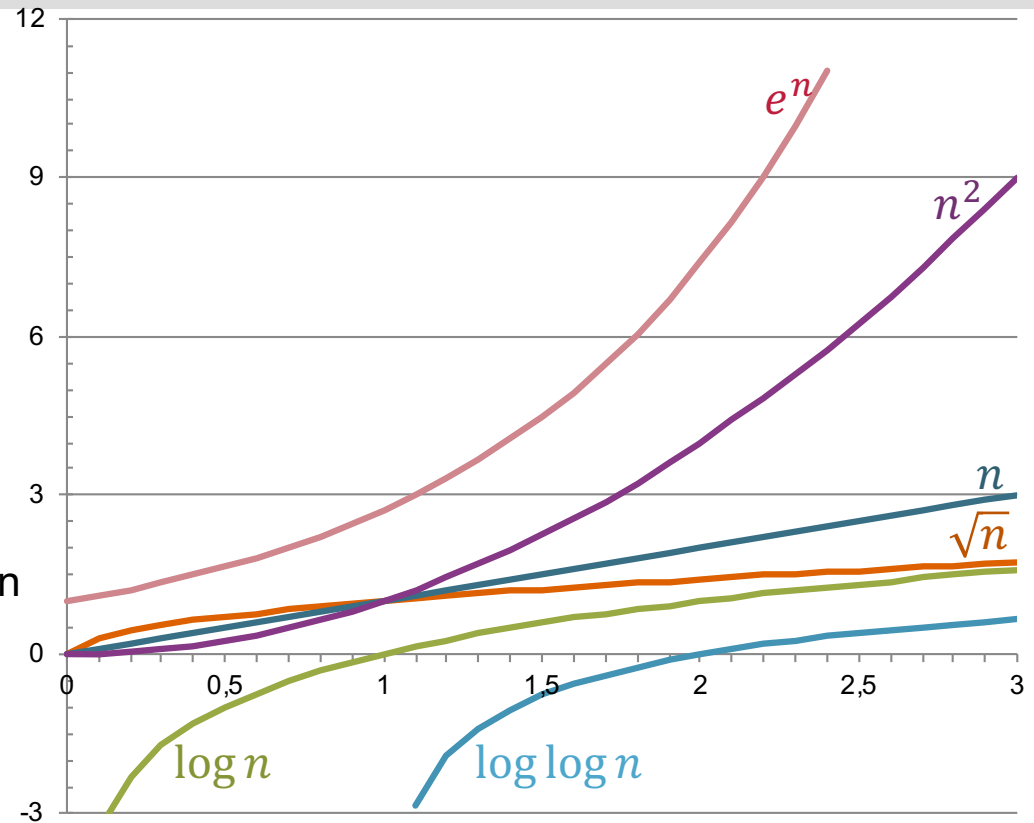
BFS/DFS, Wachstum von Funktionen

Arne Schmidt

28.11.2019

Heute

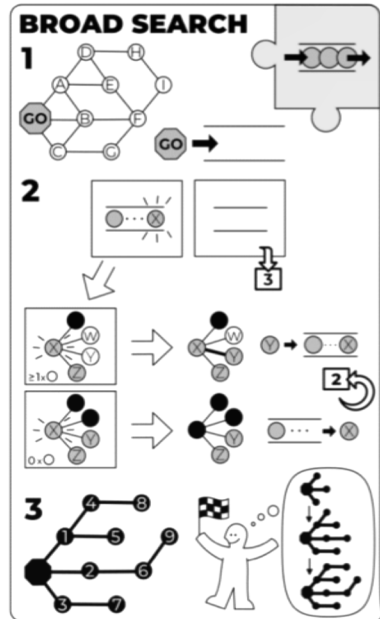
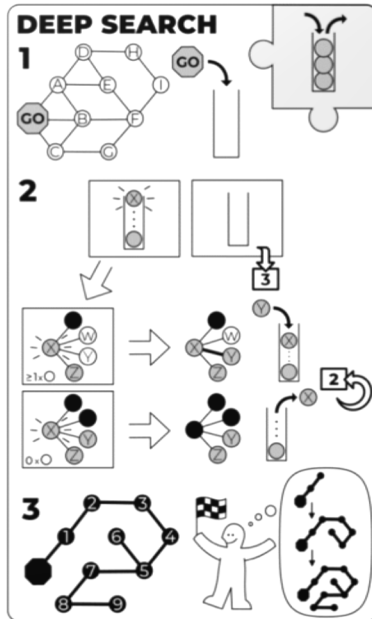
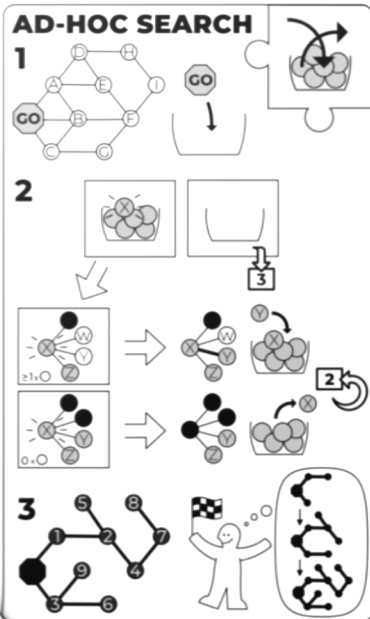
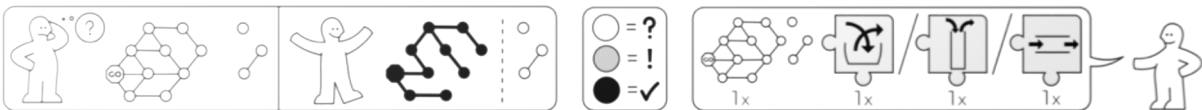
- Breiten- und Tiefensuche
- Wachstum von Funktionen
- Größe des Inputs
- Definitionen (O-Notation)
- Relationen zwischen Laufzeitklassen
- Beispiele für Laufzeitklassen



GRÄPH SKÄN

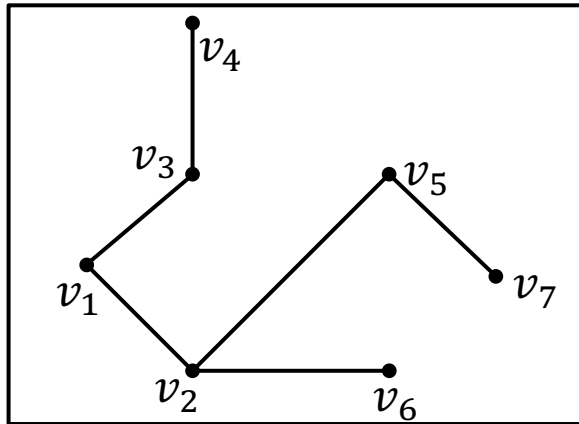
idea-instructions.com/graph-scan/
v1.2, CC by-nc-sa 4.0

IDEA

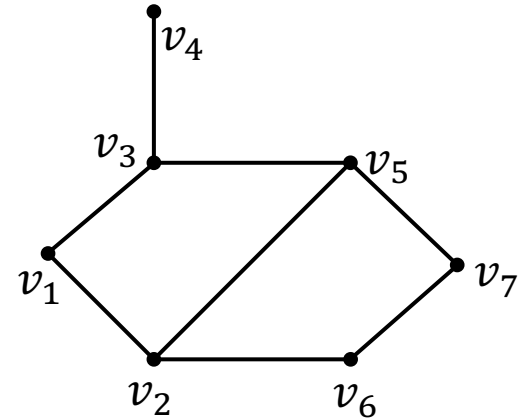


Breitensuche

Benutze Warteschlange
Prinzip: First-in-first-out



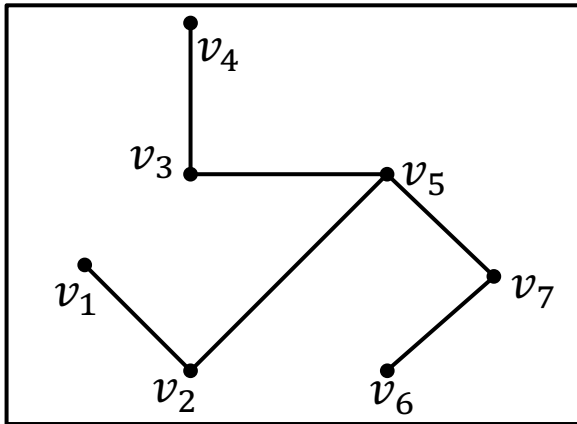
v_1
 v_1, v_2
 v_1, v_2, v_3
 v_2, v_3
 v_2, v_3, v_5
 v_2, v_3, v_5, v_6
 v_3, v_5, v_6
 v_3, v_5, v_6, v_4
 v_5, v_6, v_4
 v_5, v_6, v_4, v_7
 v_6, v_4, v_7
 v_4, v_7
 v_7
 \emptyset



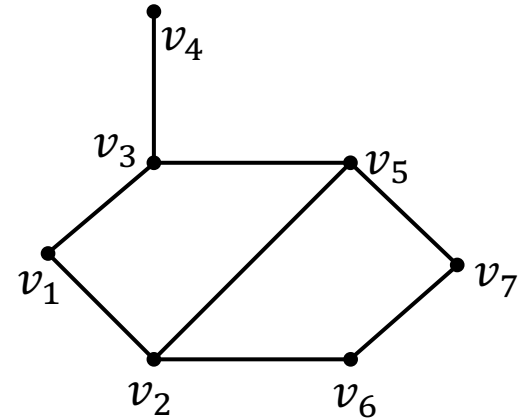
Tiefensuche

Benutze Stapel

Prinzip: Last-in-first-out



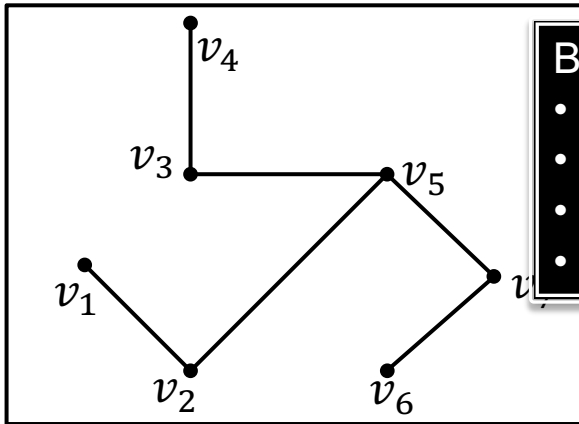
v_1
 v_1, v_2
 v_1, v_2, v_5
 v_1, v_2, v_5, v_3
 v_1, v_2, v_5, v_3, v_4
 v_1, v_2, v_5, v_3
 v_1, v_2, v_5
 v_1, v_2, v_5, v_7
 v_1, v_2, v_5, v_7, v_6
 v_1, v_2, v_5, v_7
 v_1, v_2, v_5
 v_1, v_2
 v_1
 \emptyset



Tiefensuche

Benutze Stapel

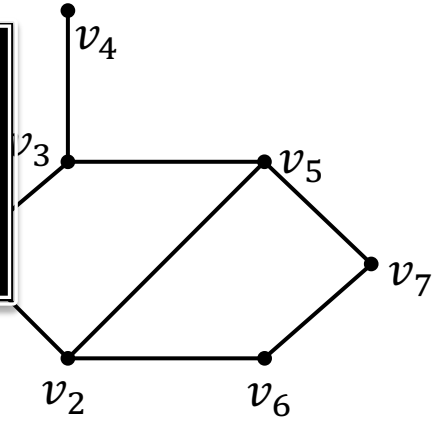
Prinzip: Last-in-first-out



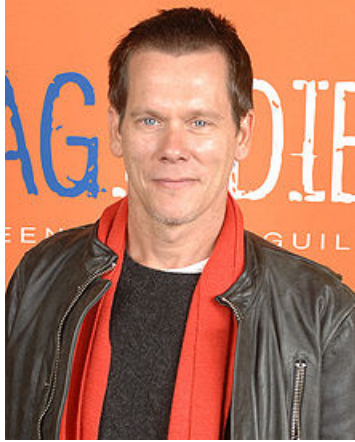
- Beliebte Fehler:**
- Leere Menge am Ende vergessen
 - Nicht jede Änderung angegeben
 - Suchbaum nicht angegeben
 - Nicht den kleinsten Index beachtet

v_1
 v_1, v_2
 v_1, v_2, v_5
 v_1, v_2, v_5, v_3
 $'' '' '' '' ''$

v_1, v_2, v_5, v_7
 v_1, v_2, v_5
 v_1, v_2
 v_1
 \emptyset



Six Degrees of Kevin Bacon

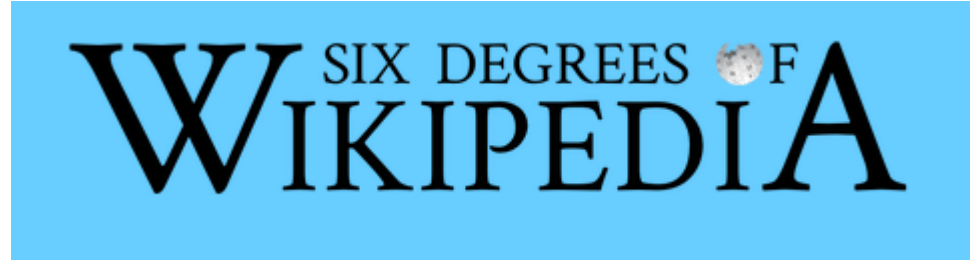


Finde kürzesten Weg von einem
Schauspieler über Filme zu Kevin Bacon

Ungerichteter Graph

Hin- und Rückweg sind gleich lang

vs. Six Degrees of Wikipedia

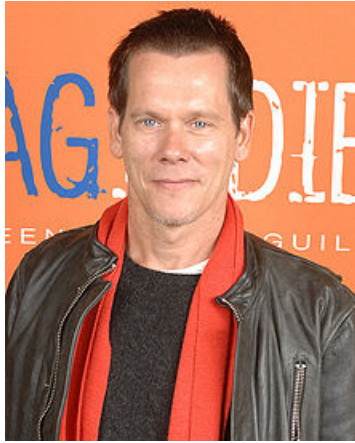


Finde kürzesten Weg von einer Seite
über enthaltene Links zu einer anderen

Gerichteter Graph

Hin- und Rückweg sind verschieden lang

Six Degrees of Kevin Bacon

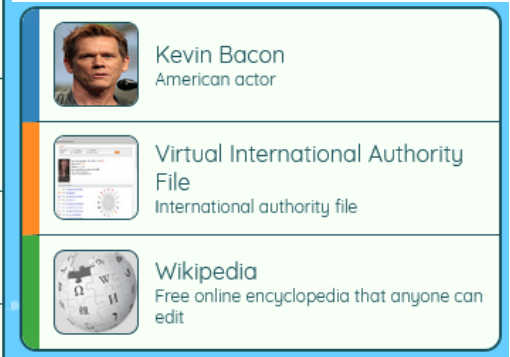
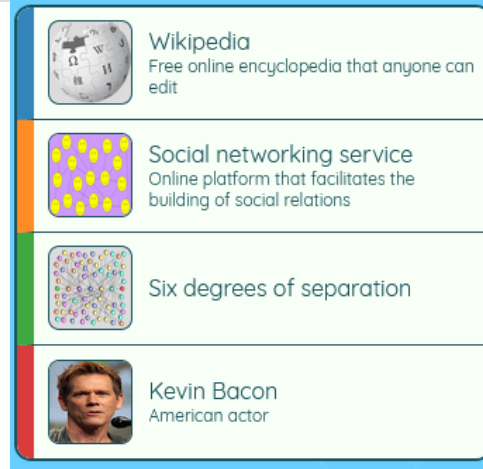


Finde kürzesten Weg von einem
Schauspieler über Filme zu Kevin Bacon

Ungerichteter Graph

Hin- und Rückweg sind gleich lang

vs. Six Degrees of Wikipedia



Finde kürzesten Weg von einer Seite
über enthaltene Links zu einer anderen

Gerichteter Graph

Hin- und Rückweg sind verschieden lang

Weitere Problem mit BFS/DFS – Eindeutigkeitsproblem

Gegeben: Graph G und Knoten v

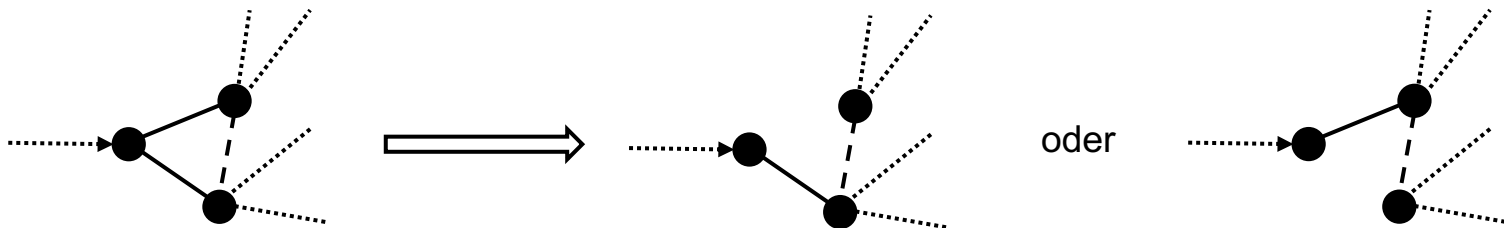
Frage: Ist der BFS(/DFS)-Baum von G eindeutig, wenn man bei v startet?

Satz: DFS-Baum ist genau dann eindeutig, wenn G ein Baum ist.

Beweis:

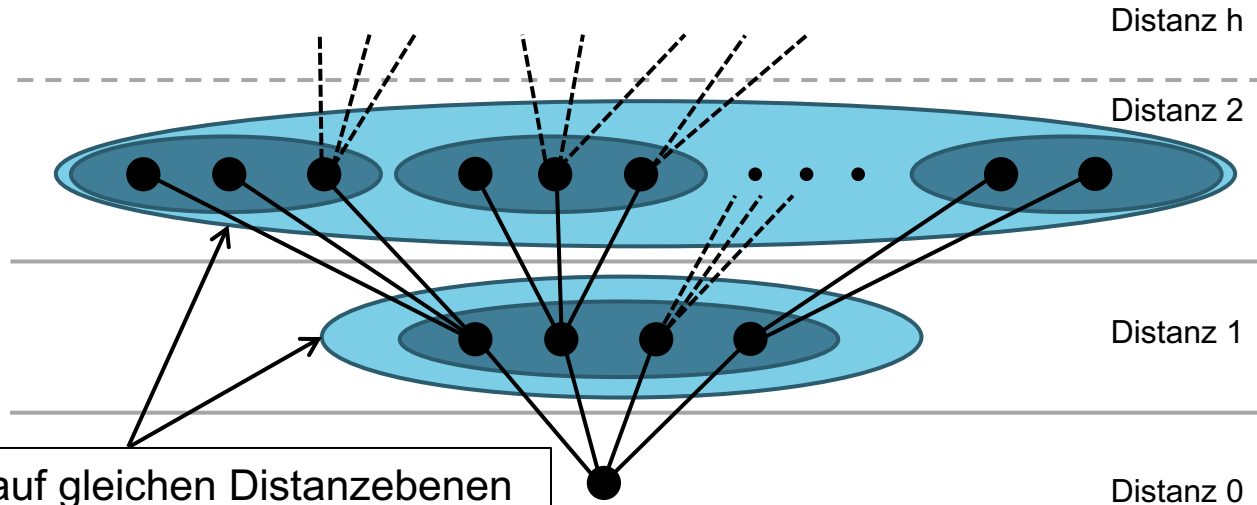
„ \Rightarrow “: Angenommen G besitzt einen Kreis. Dann können wir bei Erreichen des Kreises den Baum links oder recht herum aufbauen. Der DFS-Baum ist nicht eindeutig. Widerspruch!

„ \Leftarrow “: Wenn G ein Baum ist, ist auch der DFS-Baum ebendieser Graph. \square



Eindeutiger BFS-Baum

Satz: Der BFS-Baum ist genau dann eindeutig, wenn der kürzeste Pfad von v zu allen anderen Knoten eindeutig ist.



Alle Knoten auf gleichen Distanzebenen dürfen miteinander verbunden sein!

Labyrinth



Die Reise ins Labyrinth, 1986

<https://fictionmachine.com/2015/07/03/everything-ive-done-ive-done-for-you-labyrinth-1985/>

Zeit für ein Labyrinth

Wie oft muss man durch die Gänge des Labyrinths laufen, wenn man...

1. BFS, oder
2. DFS nutzt?

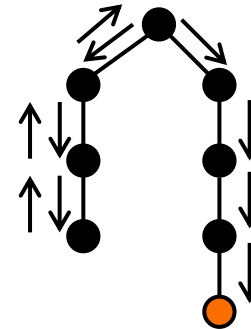
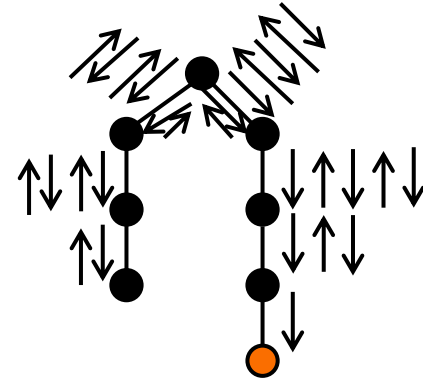
DFS schafft das schneller!

Man kann zeigen:

- Jede Kante wird maximal zwei Mal benutzt.
- Man benötigt maximal $2n-1$ Schritte
- Es gibt Bäume, bei denen $2n-1$ Schritte benötigt werden

BFS:

Worst-Case:
 $\Omega(n^2)$ mal über
Kanten laufen!



Überblick

- Breiten- und Tiefensuche
- Wachstum von Funktionen
 - Größe des Inputs
 - Definitionen (O-Notation)
 - Relationen zwischen Laufzeitklassen
 - Beispiele für Laufzeitklassen

Wachstum von Funktionen / Laufzeiten

Wie lange benötigen Algorithmen?

- Absolute/Genaue Laufzeit ist nicht immer nötig
- Abschätzen: In welcher *Ordnung* wächst die Laufzeit?
- Wovon hängt die Laufzeit ab?
 - Was ist gegeben? (Zahlen, Strings, Graphen,...)
 - Wie viel ist gegeben? (Codierungsgröße)

Codierungsgröße - Zahlen

Wie können Zahlen codiert werden?

1. Unär („Bierdeckelnotation“)

2. b-adisch

- Dezimal (10-adisch)
- Binär (2-adisch)
- Allgemein: $a_m a_{m-1} \dots a_1 a_0, a_{-1} a_{-2} \dots a_{-\infty}$ wobei $n = \sum_{i=-\infty}^m a_i b^i$ und $0 \leq a < b$

Gewöhnlich für Zahlen:
Binäre Darstellung, d.h.
eine Zahl belegt $\log_2 n$ bits.

Beispiele:

| Dezimal | Unär | Binär | Oktal | Hexadezimal |
|---------|------|-------|-------|-------------|
| 27 | | 11011 | 33 | 1B |

Codierungsgröße - Graphen

Wie kann man Graphen speichern?

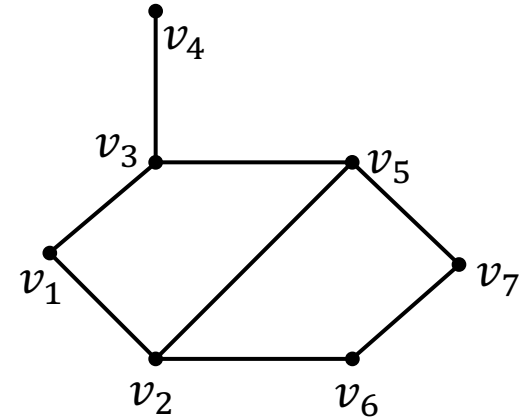
Adjazenzmatrix:

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Größe: $|V|^2$ bits

Adjazenzliste:

$v_1: v_2, v_3$
 $v_2: v_1, v_5, v_6$
 $v_3: v_1, v_4, v_5$
 $v_4: v_3$
 $v_5: v_2, v_3, v_7$
 $v_6: v_2, v_7$
 $v_7: v_5, v_6$



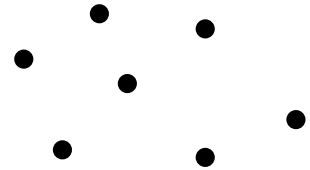
$\approx (|V| + |E|) \cdot \log|V|$ bits

Codierungsgröße - Sonstige

Was gibt es noch?

- Zeichenketten/Strings S : $\approx |S| \cdot \log N$
für N mögliche Zeichen.
- Punktmenge P in d Dimensionen: $\approx d \cdot |P| \cdot \log N$,
wobei N die größte Koordinate ist.
- $n \times m$ -Matrizen: $\approx n \cdot m \cdot \log(N)$,
wobei N der größtmögliche Wert ist.

(>o.o)>Das_ist_ein_String!!¥(°o°)¥



$$\begin{pmatrix} 5 & 12 & 1 \\ 6 & 22 & 5 \\ 0 & 42 & 21 \end{pmatrix}$$

Überblick

- Breiten- und Tiefensuche
- Wachstum von Funktionen
 - Größe des Inputs
 - Definitionen (O-Notation)
 - Relationen zwischen Laufzeitklassen
 - Beispiele für Laufzeitklassen

Wachstum von Funktionen

Betrachte Laufzeit als Funktion $T: \mathbb{N} \rightarrow \mathbb{R}^+$

Verschiedene Systeme liefern verschiedene Laufzeiten, z.B. $T_1(n)$ und $T_2(n)$.

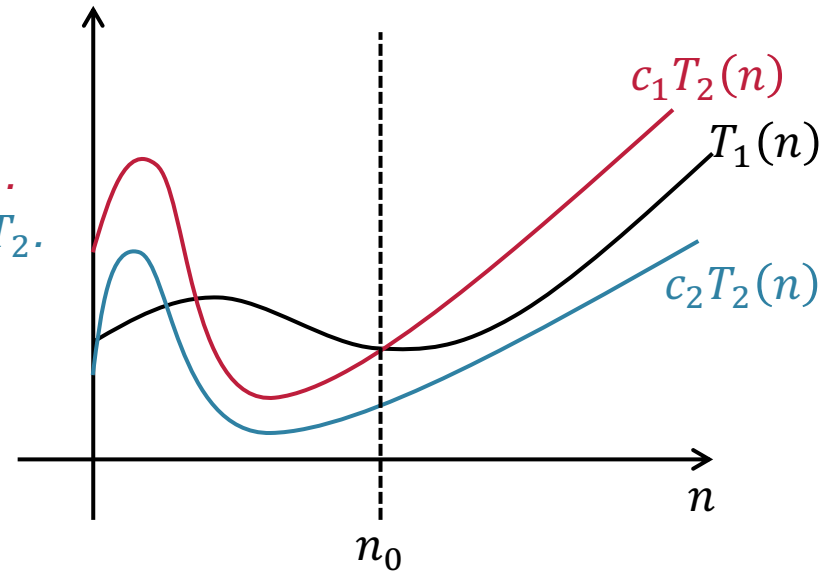
Aber:

$T_1(n)$ und $T_2(n)$ wachsen “ähnlich schnell”

$T_1(n)$ wächst höchstens c_1 mal so schnell wie T_2 .

$T_1(n)$ wächst mindestens c_2 mal so schnell wie T_2 .

Das gilt ggf. erst ab einem bestimmten Punkt,
nämlich n_0



Landau-Symbole

Das motiviert folgende Definitionen:

O -Notation:

Es gibt Konstanten $n_0 \in \mathbb{N}$ und $c_1 \in \mathbb{R}^+$, sodass für alle $n \geq n_0$ gilt:

$$0 \leq T_1(n) \leq c_1 T_2(n) \Leftrightarrow T_1(n) \in O(T_2(n))$$

Ω -Notation:

Es gibt Konstanten $n_0 \in \mathbb{N}$ und $c_2 \in \mathbb{R}^+$, sodass für alle $n \geq n_0$ gilt:

$$T_1(n) \geq c_2 T_2(n) \geq 0 \Leftrightarrow T_1(n) \in \Omega(T_2(n))$$

Θ -Notation:

Es gibt Konstanten $n_0 \in \mathbb{N}$ und $c_1, c_2 \in \mathbb{R}^+$, sodass für alle $n \geq n_0$ gilt:

$$0 \leq c_2 T_2(n) \leq T_1(n) \leq c_1 T_2(n) \Leftrightarrow T_1(n) \in \Theta(T_2(n))$$

Achtung:
 O , Ω und Θ sind Mengen!

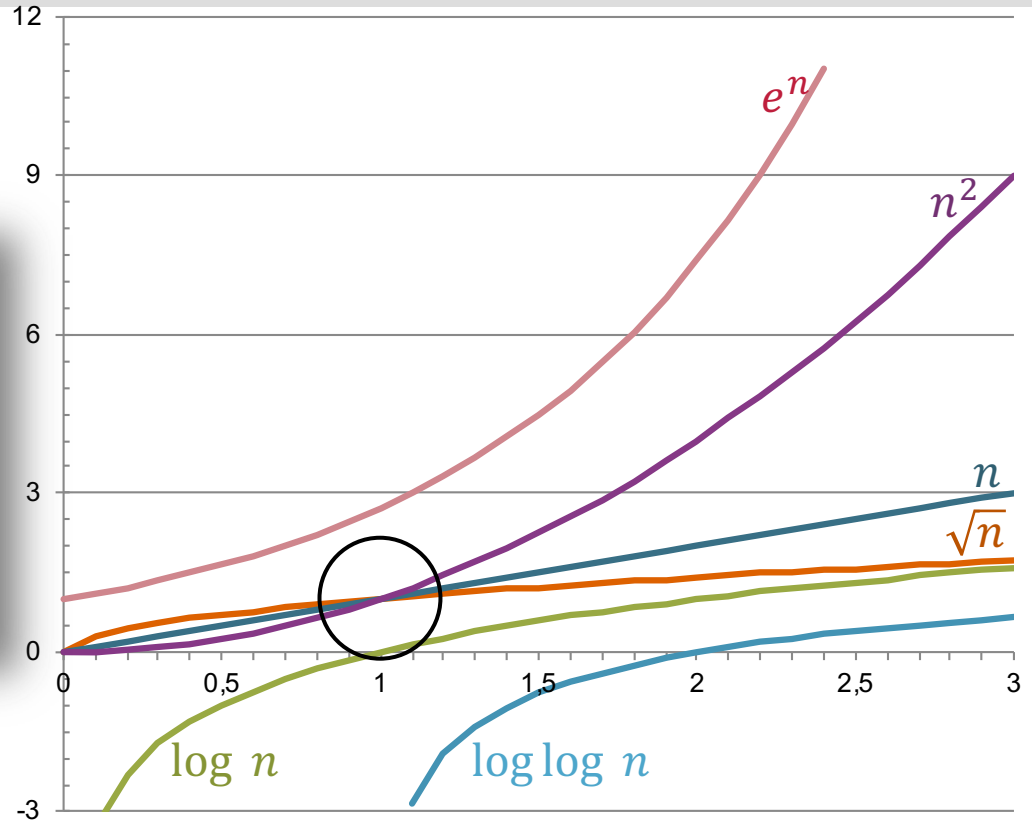
Beispiel Funktionen

~~Zeige: $n \in O(n^2)$~~

~~Mit $c = 1$ und $n_0 = 1$ gilt die Aussage wie rechts zu sehen.~~



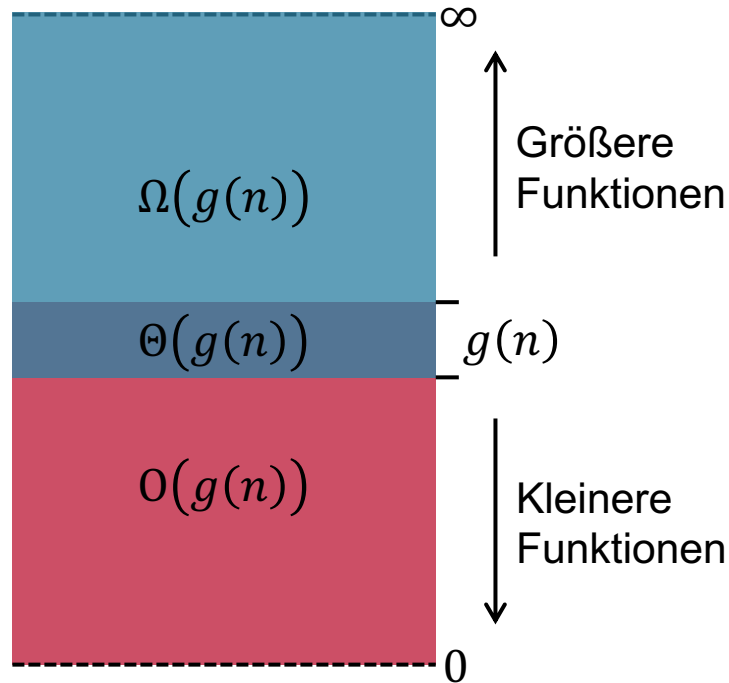
Vorsicht: Es muss eine Aussage
bewiesen/widerlegt werden



Überblick

- Breiten- und Tiefensuche
- Wachstum von Funktionen
 - Größe des Inputs
 - Definitionen (O-Notation)
 - Relationen zwischen Laufzeitklassen
 - Beispiele für Laufzeitklassen

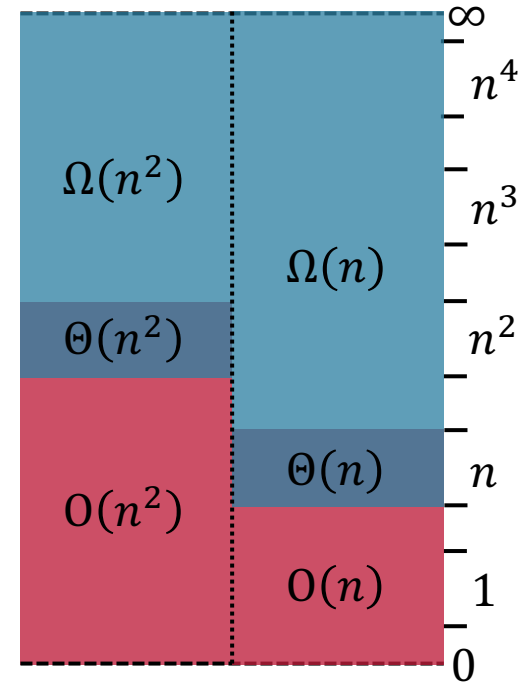
Relationen zwischen Klassen - Eine graphische Darstellung



Wir können sogar Klassen miteinander vergleichen.
Beispiel:

$$\Theta(n^2) \not\subseteq \Omega(n)$$

Zum Merken:
Wird die Funktion größer,
wächst der O -Bereich und
der Ω -Bereich schrumpft.



Relationen zwischen Klassen – Eine Tabelle

| Bedingung | Klasse | | | |
|---|----------------|-----------------|-----------------|-----------------|
| | Klasse | $O(g(n))$ | $\Theta(g(n))$ | $\Omega(g(n))$ |
| $f(n) \in o(g(n))$ $(o(g(n)) := O(g(n)) \setminus \Theta(g(n)))$ "Klein-o-Notation" | $O(f(n))$ | $\not\subseteq$ | X | X |
| | $\Theta(f(n))$ | $\not\subseteq$ | X | X |
| | $\Omega(f(n))$ | X | $\not\supseteq$ | $\not\supseteq$ |
| $f(n) \in \Theta(g(n))$ | $O(f(n))$ | = | $\not\supseteq$ | X |
| | $\Theta(f(n))$ | \subseteq | = | \subseteq |
| | $\Omega(f(n))$ | X | $\not\supseteq$ | = |
| $f(n) \in \omega(g(n))$ $(\omega(g(n)) := \Omega(g(n)) \setminus \Theta(g(n)))$ "Klein- ω -Notation" | $O(f(n))$ | $\not\supseteq$ | $\not\supseteq$ | X |
| | $\Theta(f(n))$ | X | X | \subseteq |
| | $\Omega(f(n))$ | X | X | \subseteq |

Heute

- Breiten- und Tiefensuche
- Wachstum von Funktionen
 - Größe des Inputs
 - Definitionen (O-Notation)
 - Relationen zwischen Laufzeitklassen
 - Beispiele für Laufzeitklassen

Beispiel 1

Zeige $4n^2 + 12n - 15 \in \Theta(n^2)$.

D.h.: Bestimme n_0, c_1, c_2 , sodass für alle $n \geq n_0$ gilt:

$$0 \leq c_1 \cdot n^2 \leq 4n^2 + 12n - 15 \leq c_2 \cdot n^2$$

1. Suche nach c_2 :

$$4n^2 + 12n - 15$$

2. Suche nach c_1 :

$$4n^2 + 12n - 15$$

Beide Ungleichung gelten ab $n_0 = 4$. Also $c_1 = 3, c_2 = 16$ und $n_0 = 4$.

Beispiel 2

Zeige oder widerlege: $2^n \in \Theta(3^n)$

Zunächst: $2^n \in O(3^n)$, denn $2^n \leq (2 + 1)^n = 3^n$.

Aber: $2^n \notin \Omega(3^n)$!

Ansonsten gäbe es eine Konstante c_1 mit

$$2^n \geq c_1 \cdot 3^n, \text{ also } \frac{2^n}{3^n} \geq c_1$$

Aber: $\frac{2^n}{3^n} = \left(\frac{2}{3}\right)^n$ und $\lim_{n \rightarrow \infty} \left(\frac{2}{3}\right)^n = 0$, d.h. dieses c_1 kann nicht existieren!

\Rightarrow Aussage widerlegt.

Beispiel 3

Satz 3.1: Für jedes $c > 0$ gibt es ein n_0 , sodass für alle $n \geq n_0$ gilt: $\log_2 n \leq c \cdot n$

Beweisidee: Zeige, dass $\lim_{n \rightarrow \infty} \frac{\log_2 n}{n} = 0$, d.h. für wachsendes n kommen wir beliebig nah an 0 heran. D.h. wir können n_0 so wählen, dass $\frac{\log_2 n}{n} \leq c$ für alle $n \geq n_0$ gilt.

Beispiel 3

Satz 3.1: Für jedes $c > 0$ gibt es ein n_0 , sodass für alle $n \geq n_0$ gilt: $\log_2 n \leq c \cdot n$

Satz 3.2: Seien $a, b \in \mathbb{R}^+$. Dann gilt $\log_2^a n \in O(n^b)$.

Beweis: Wählen wir die Konstante $c = 1$. Dann soll gelten:

$$\begin{aligned} & \log_2^a n \leq n^b \\ \Leftrightarrow & \log_2 \log_2^a n \leq \log_2 n^b \\ \Leftrightarrow & a \cdot \log_2 \log_2 n \leq b \cdot \log_2 n \\ m := \log_2 n & \Leftrightarrow \log_2 m \leq \frac{b}{a} m \end{aligned}$$

Da $a, b \in \mathbb{R}^+$ ist auch $\frac{b}{a} \in \mathbb{R}^+$. Nach Satz 3.1 ist die letzte Ungleichung ab einem n_0 wahr. Da wir Äquivalenzumformungen benutzt haben, können wir die Lösungskette von unten nach oben gehen.