Technische Universität Braunschweig

Institute of Operating Systems and Computer Networks

Algorithmen und Datenstrukturen
Große Übung #3

Phillip Keldenich, Arne Schmidt

24.11.2016

Heute: Komplexität von Algorithmen

Komplexität

Was ist Komplexität?

- Laufzeit
- Speicherbedarf

Wie berechne ich das?

Wovon ist das abhängig?

- Größe des Inputs $\Rightarrow$ Codierung?

Input Codierung - Zahlen

Unär (Anzahl Striche entspricht der Zahl) oder
b-adische Zahl:

- Dezimal: 10-adisch
- Binär: 2-adisch
- Allgemein: $a_m \dots a_1 a_0, a_{-1} \dots a_{-\infty}$ wobei $n = \sum_{i=-\infty}^{m} a_i \cdot b^i$

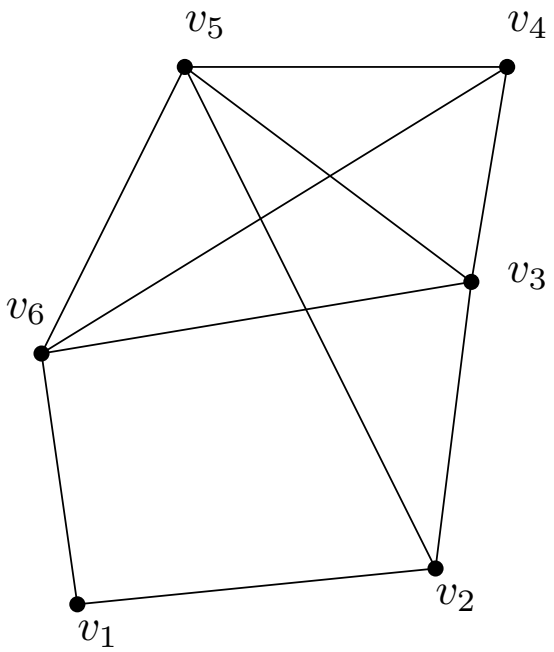
Beispiele für die Zahl 27:

- unär: |||||||||||||||||||||||||||
- binär: 11011

Als Input für ein Programm betrachten wir die binäre Darstellung. Also ist die Größe einer Zahl n als Input $\log_2 n$.

Input Codierung - Graphen

Wie codieren wir Graphen?
Zum Beispiel mit einer Adjazenzmatrix:



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

Größe Adjazenzmatrix: $|V|^2$

Input Codierung - Sonstiges

- Graph (Inzidenzmatrix): $|V| \cdot |E|$
- Graph (Adjazenzliste): $|V| + |E|$
- Zeichenketten/Strings S: $|S|$
- Punktmenge P in d Dimensionen: $d|P|$
- $n \times m$ Matrizen: $n \cdot m$

Laufzeitanalyse

Wie analysiere ich die Laufzeit eines Algorithmus?
Laufzeit als Funktion $T : \mathbb{N} \to \mathbb{N}$. Aber:

- unterschiedliche Systeme (Rechenschieber, Amiga, High-End PC)
- unterschiedliche Laufzeiten $(T_1(n), T_2(n), T_3(n))$

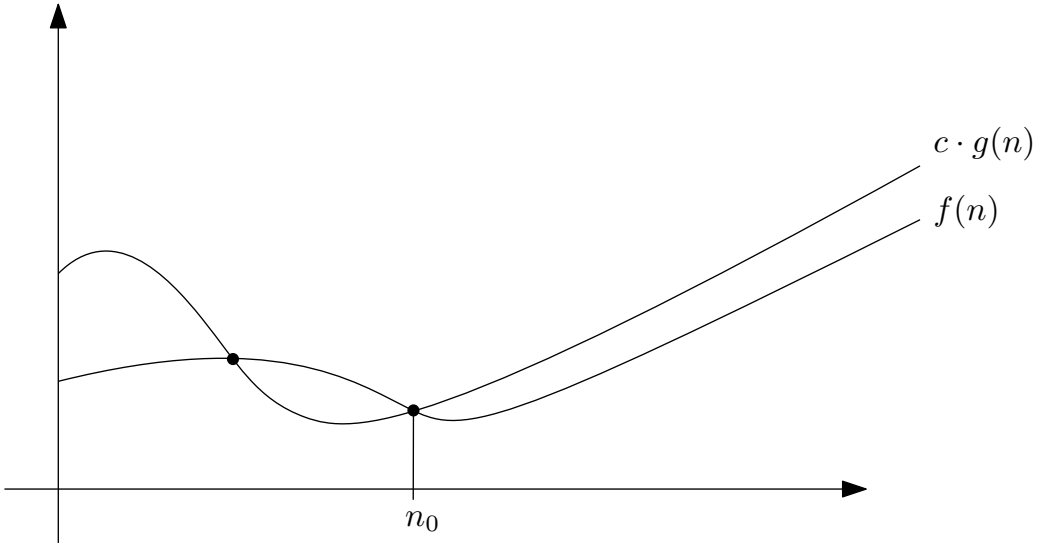
Am besten: System unabhängig!
$T_1(n)$, $T_2(n)$ und $T_3(n)$ wachsen „ähnlich schnell“: $T_1(n)$ ist c_1 mal langsamer als $T_2(n)$, also $T_1(n) \leq c_1 \cdot T_2(n)$.

Das heißt, $T_2(n)$ wächst mindestens so schnell wie $T_2(n)$!

O-, Ω- und Θ-Notation

Wichtig: $T_1(n)$ kann größer als $c_1 \cdot T_2(n)$ sein, wenn n „klein“ist.

Interessant sind große n: Ab n_0 gilt **immer** $T_1(n) \leq c_1 \cdot T_2(n)$



Formal: $\exists n_0 \in \mathbb{N}, c_1 \in \mathbb{R}^+ : \forall n \geq n_0 : T_1(n) \leq c_1 \cdot T_2(n)$.
D.h. $T_1(n) \in O(T_2(n))$.
Achtung: Zeichnung reicht nicht, um $T_1(n) \in O(T_2(n))$ zu beweisen!

O-, Ω- und Θ-Notation

Neben der O-Notation gibt es auch die Ω- und Θ-Notation.
Da $T_1(n) \leq c_1 \cdot T_2(n)$ gilt, gilt auch $\frac{T_1(n)}{c_1} \leq T_2(n)$, also $T_2(n) \in \Omega(T_1(n))$.
Das heißt, $T_1(n)$ wächst höchstens so schnell wie $T_2(n)$!

Existiert zusätzlich $c_2 > 0$, sodass $c_2 \cdot T_2(n) \leq T_1(n) \leq c_1 \cdot T_2(n)$, dann ist $T_1 \in \Theta(T_2(n))$. Das heißt, beide Funktionen sind asymptotisch gleich!

Rechenregeln

Es ist gut, gewisse „Rechenregeln“ zu kennen:

- $O(g(n)) + O(f(n)) = \begin{cases} O(g(n)), \text{ falls } f(n) \in O(g(n)) \\ O(f(n)), \text{ falls } g(n) \in O(f(n)) \end{cases}$
- $O(g(n)) \cdot O(f(n)) = O(g(n) \cdot f(n))$
- $a \cdot O(f(n)) = O(f(n))$, für $a \in \mathbb{R}^+$
- $O(O(f(n))) = O(f(n))$

Für Ω-Notation sind die Regeln ähnlich.
Was ist mit Subtraktion? $\rightarrow$ Geht nicht immer!

Ein paar Beispiele (1)

$4n^2 + 12n - 15 \in \Theta(n^2)$
Finde $c_1, c_2 \in \mathbb{R}^+$ und $n_0 \in \mathbb{N}$, sodass $\forall n \geq n_0$ gilt:

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

$4n^2 + 12n - 15 \leq 16n^2 - 15 \leq 16n^2$, für $n_0 \geq 1$

$4n^2 + 12n - 15 \geq 4n^2 - 15 \overset{n_0 \geq 4}{\geq} 3n^2$, für $n_0 \geq 4$

Wir wählen also $c_1 = 3, c_2 = 16$ und $n_0 = 4$.

Ein paar Beispiele (2)

Wenn $f(n) \in O(g(n))$ und $f(n) \in O(h(n))$, dann ist $g(n) \in O(h(n))$?

Diese Aussage gilt nicht! Widerlegbar durch ein Beispiel:
Wähle $f(n) = h(n) = n$ und $g(n) = n^2$

Es gilt $f(n) = n \leq n^2 = g(n)$ und $f(n) = n \leq n = h(n)$.
Ist $g(n) \in O(h(n))$?

Nach Definition muss gelten $g(n)/h(n) \leq c$ für alle $n \geq n_0$. Aber $\frac{n^2}{n} = n$ ist nicht konstant. Daher: $g(n) \notin O(h(n))$.

Ein paar Beispiele (3)

$\forall a, b \in \mathbb{R}^+ : \log^a(n) \in O(n^b)$
Man kann zeigen: $\forall c \in \mathbb{R}^+ : \exists m_0 \in \mathbb{N} : \forall m \geq m_0 : \log(m) \leq c \cdot m$.
Kurz: $\log n \in o(n)$
Außerdem: Logarithmus ist streng monoton wachsend.

$$\log^a(n) \leq n^b \Leftrightarrow \log(\log^a(n)) \leq \log(n^b) \Leftrightarrow \log(\log(n)) \leq \frac{b}{a}\log(n)$$

$$\overset{m := \log(n)}{\Longleftrightarrow} \log(m) \leq \frac{b}{a} m$$

Da $\log(m) \leq \frac{b}{a} m$ ab einem n_0 gilt, gilt auch $\log^a(n) \leq n^b$ ab n_0 mit $c = \frac{b}{a}$.

Laufzeitanalyse eines Algorithmus (1)

```
1:  function KOMPLEX(G = (V, E))
2:      for v in V do
3:          Output 'v: '
4:          for w in V do
5:              if {v, w} ∈ E then
6:                  Output 'w, '
7:              end if
8:          end for
9:      end for
10: end function
```

Laufzeit von KOMPLEX:

- Zeilen 5+6 dauern: $O(1) + O(1) = O(1)$.
- For-Schleife Zeile 4 wird $O(n)$-mal wiederholt.
- Damit dauern Zeilen 4-7: $O(1) \cdot O(n) = O(n)$.
- For-Schleife Zeile 2 wird $O(n)$-mal wiederholt.
- Zeilen 2-9 dauern: $O(n) \cdot O(n) = O(n^2)$.

Laufzeitanalyse eines Algorithmus (2)

```
1:  function PrimFaktor(n)
2:      i ← 2
3:      while i ≤ n do
4:          if n mod i = 0 then
5:              n ← n/i
6:              Output i
7:          else
8:              i ← i + 1
9:          end if
10:     end while
11: end function
```

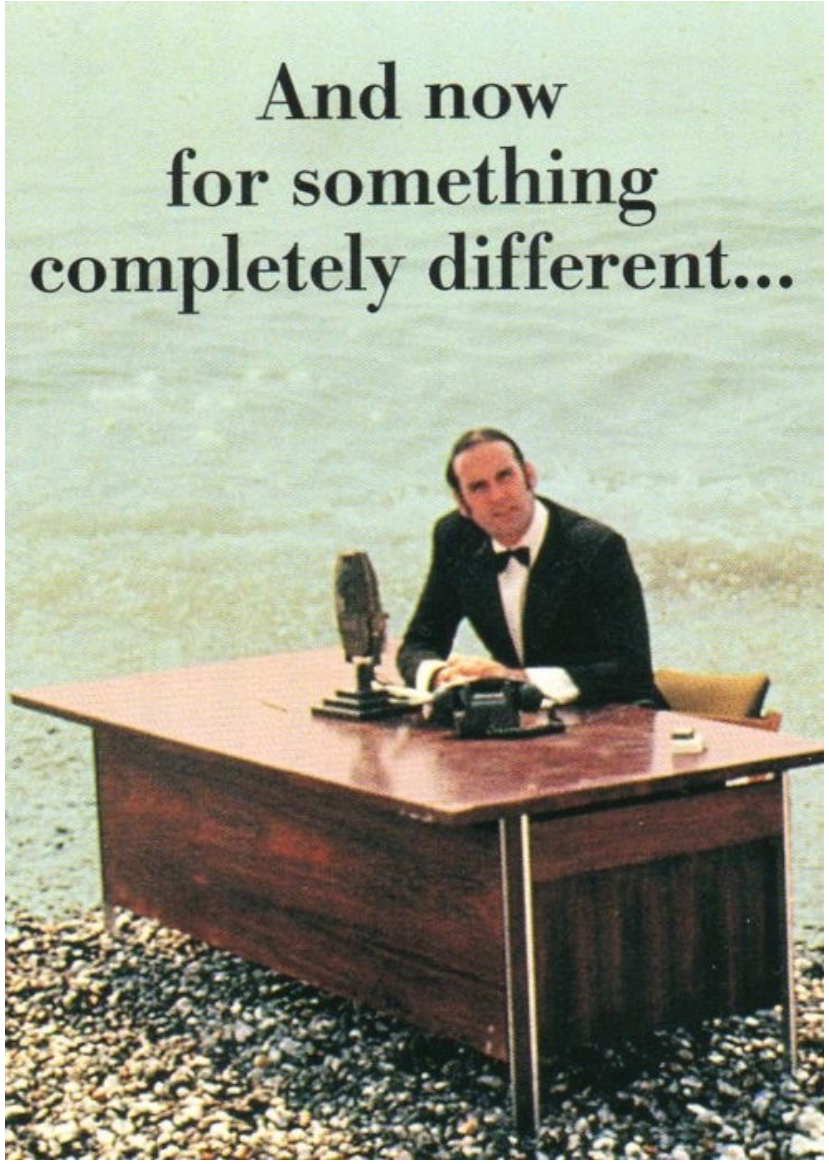
Laufzeit von PrimFaktor:

- Zeilen 4 bis 9 dauern: $O(1)$.
- While-Schleife Zeile 3 wird $O(n)$-mal wiederholt.
- Damit dauern Zeilen 2-10: $O(1) \cdot O(n) = O(n)$.

Das sieht linear aus, ist es aber nicht! Inputgröße ist $m := \log(n)$. Laufzeit ist also: $O(n) = O(2^{\log n}) = O(2^m)$. Das ist exponentiell!

Welche Funktionen liegen in $O/\Omega/\Theta$?

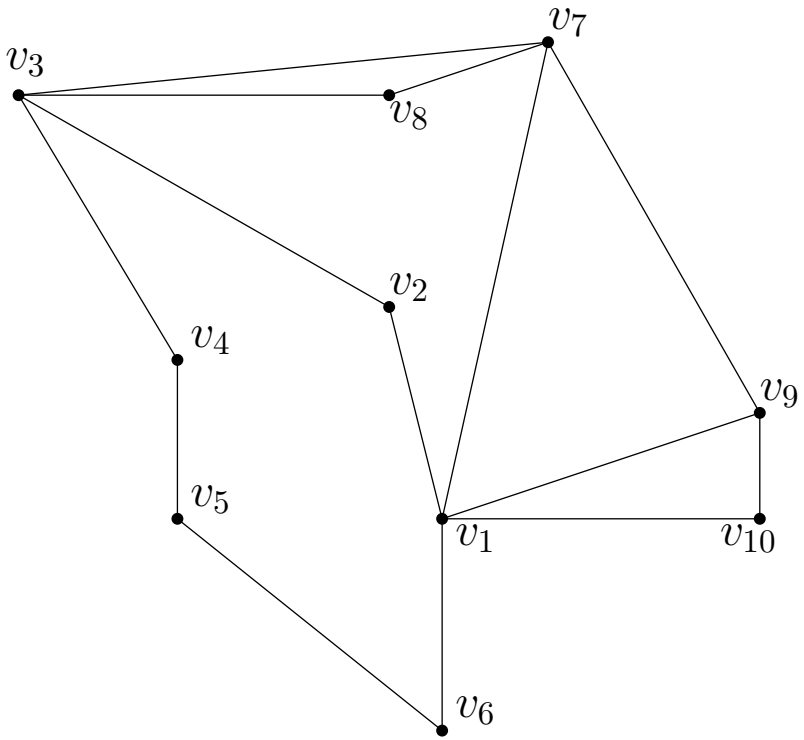
$f(n)$	$\Omega(1)$	$O(1)$	$\Theta(1)$	$\Omega(n)$	$O(n)$	$\Theta(n)$	$\Omega(n^2)$	$O(n^2)$	$\Theta(n^2)$
$\log(n)$	×				×			×	
$15n$	×			×	×	×		×	
$7^{154.12}$	×	×	×		×			×	
$\frac{n}{\log(n)}$	×				×			×	
$10^{-35}n^3$	×			×			×		
$15n^2 - 2^{50}n$	×			×			×	×	×


And now
for something
completely different...

Fragen zu den Hausaufgaben?

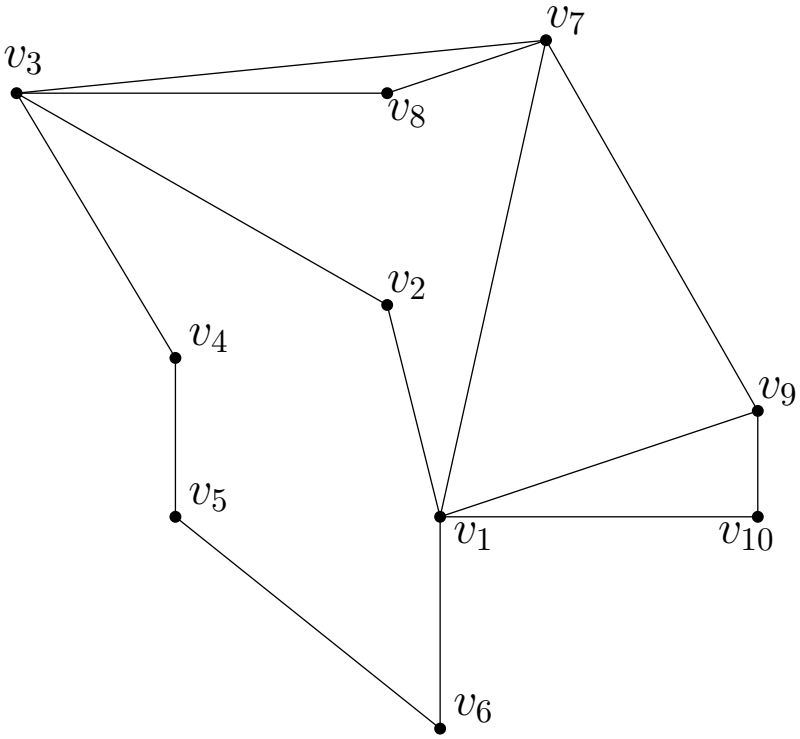
Breiten- und Tiefensuche Beispiel

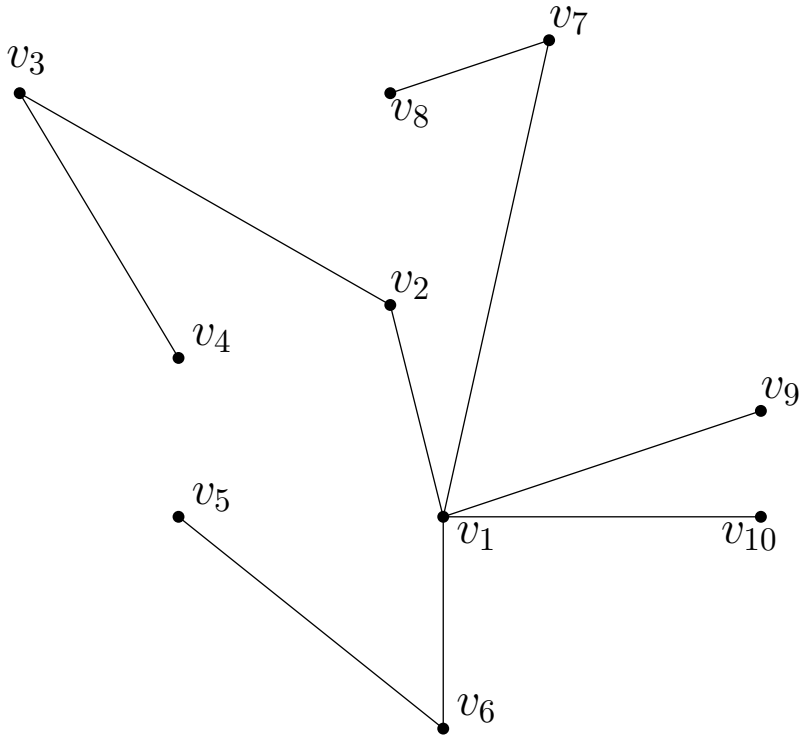
BFS



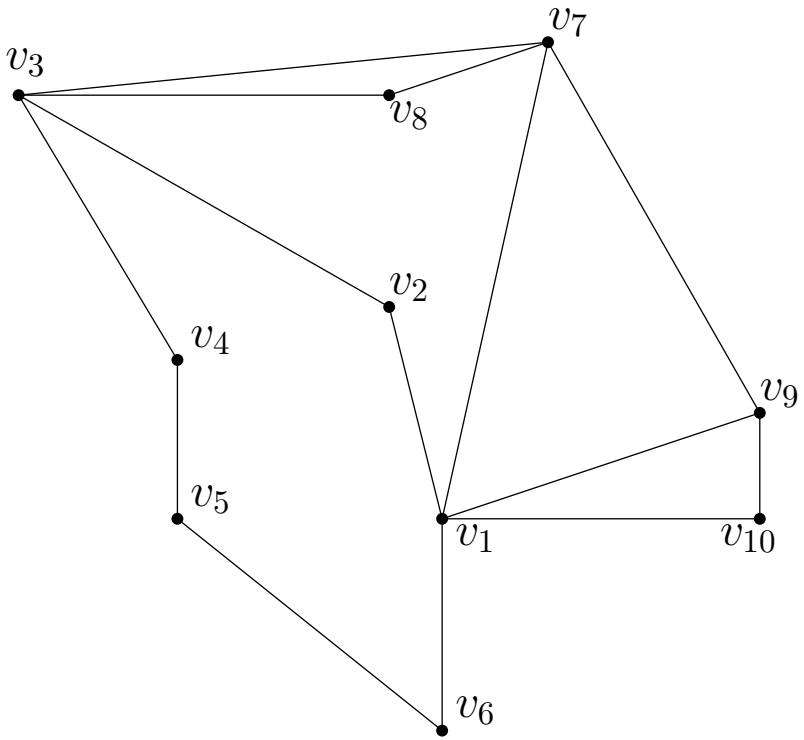
R :	v_1	$v_7, v_9, v_{10}, v_3, v_5$
	v_1, v_2	$v_7, v_9, v_{10}, v_3, v_5, v_8$
	v_1, v_2, v_6	$v_9, v_{10}, v_3, v_5, v_8$
	v_1, v_2, v_6, v_7	v_{10}, v_3, v_5, v_8
	v_1, v_2, v_6, v_7, v_9	v_3, v_5, v_8
	$v_1, v_2, v_6, v_7, v_9, v_{10}$	v_3, v_5, v_8, v_4
	$v_2, v_6, v_7, v_9, v_{10}$	v_5, v_8, v_4
	$v_2, v_6, v_7, v_9, v_{10}, v_3$	v_8, v_4
	$v_6, v_7, v_9, v_{10}, v_3$	v_4
	$v_6, v_7, v_9, v_{10}, v_3, v_5$	$\emptyset$

BFS





DFS



R :	v_1	v_1, v_2, v_3, v_7, v_8
	v_1, v_2	v_1, v_2, v_3, v_7
	v_1, v_2, v_3	v_1, v_2, v_3, v_7, v_9
	v_1, v_2, v_3, v_4	$v_1, v_2, v_3, v_7, v_9, v_{10}$
	v_1, v_2, v_3, v_4, v_5	v_1, v_2, v_3, v_7, v_9
	$v_1, v_2, v_3, v_4, v_5, v_6$	v_1, v_2, v_3, v_7
	v_1, v_2, v_3, v_4, v_5	v_1, v_2, v_3
	v_1, v_2, v_3, v_4	v_1, v_2
	v_1, v_2, v_3	v_1
	v_1, v_2, v_3, v_7	$\emptyset$

DFS