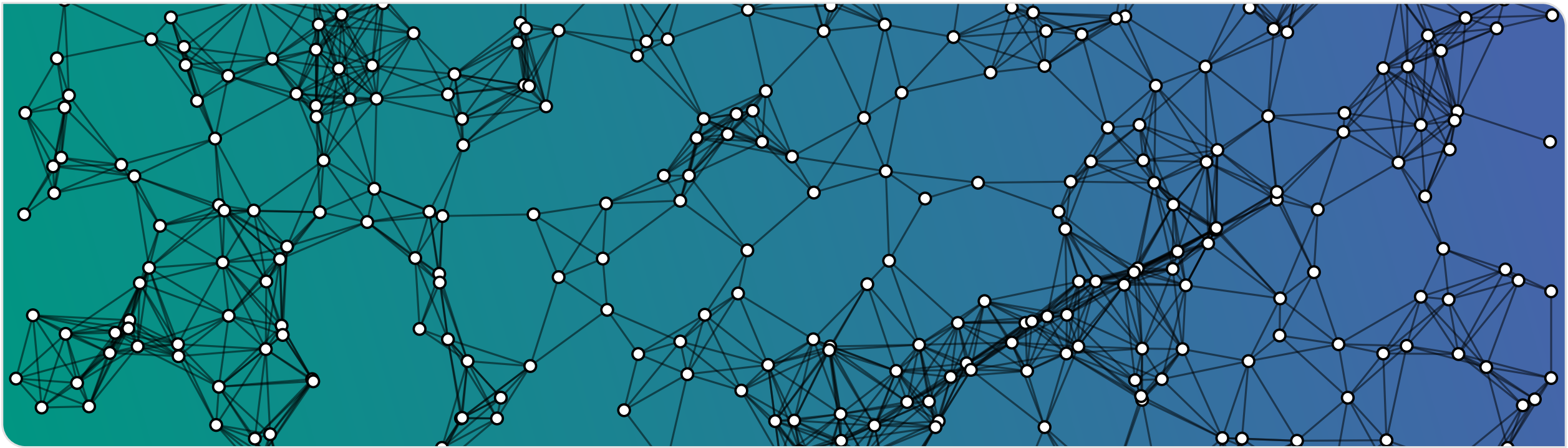
Karlsruher Institut für Technologie

Algorithmen 1

Übung 4
Graphen



Ankündigung: Nachholtutorien im August

Idee

- Wiederholung wichtiger Themen
- Üben für Prüfung

Tutoren

- Tobias Knorr
- Henriette Färber
- Jonas Seiler
- Carina Weber

Termine

August 2022

So	Mo	Di	Mi	Do	Fr	Sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

- Montags 9:45 und Mittwochs 9:45
- gesamter August

Ankündigung: Nachholtutorien im August

Idee

- Wiederholung wichtiger Themen
- Üben für Prüfung

Tutoren

- Tobias Knorr
- Henriette Färber
- Jonas Seiler
- Carina Weber

Termine

August 2022

So	Mo	Di	Mi	Do	Fr	Sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	1		

- Montags 9:45 und Mittwochs 9:45
- gesamter August

Ankündigung: Nachholtutorien im August

Idee

- Wiederholung wichtiger Themen
- Üben für Prüfung

Tutoren

- Tobias Knorr
- Henriette Färber
- Jonas Seiler
- Carina Weber

Termine

August 2022

So	Mo	Di	Mi	Do	Fr	Sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	1		

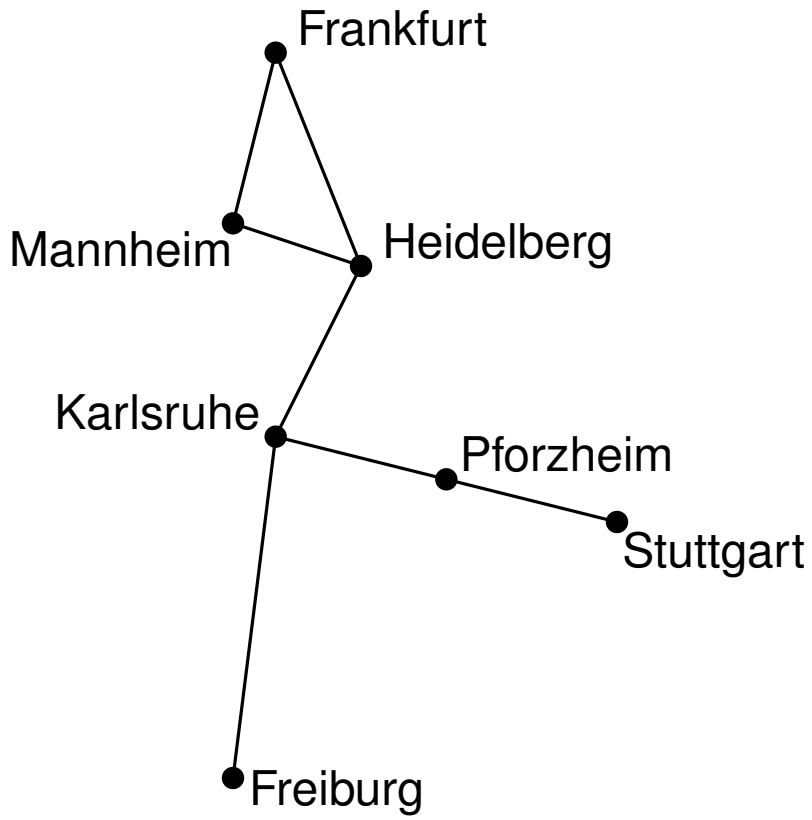
- Montags 9:45 und Mittwochs 9:45
- gesamter August

Graphen

- flexibles Tool zur Modellierung

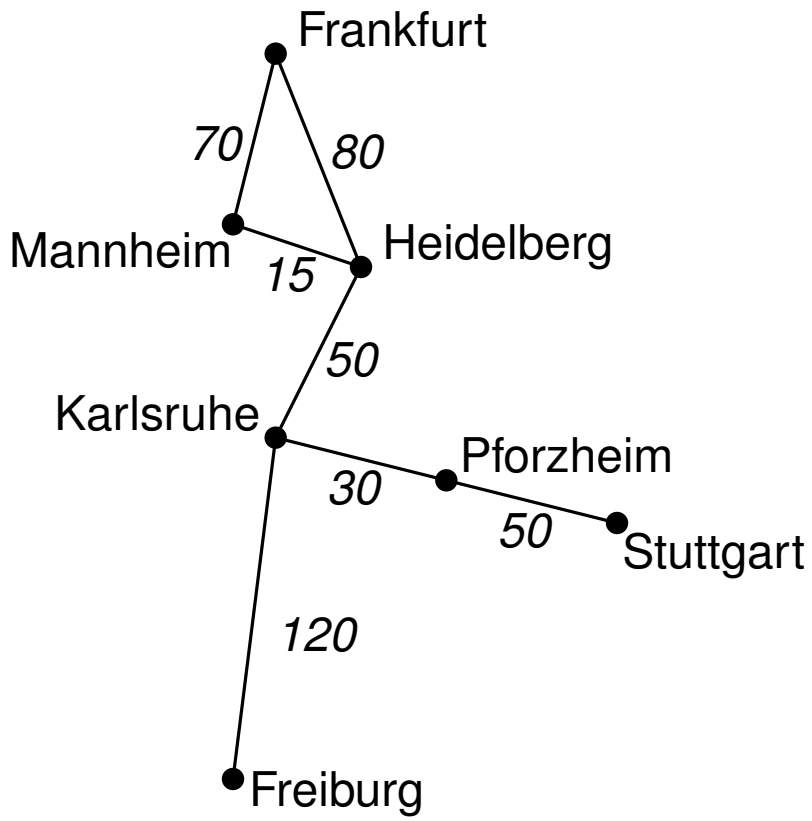
Graphen

- flexibles Tool zur Modellierung
  - Transportnetzwerke



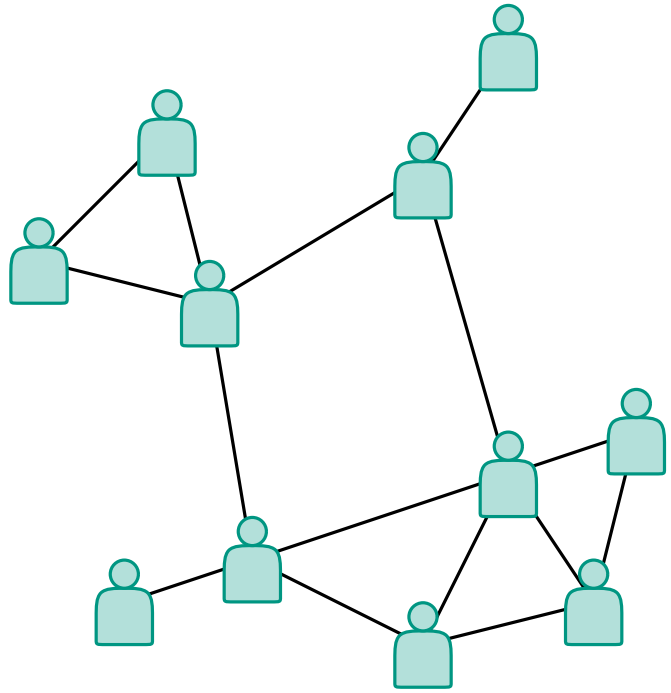
Graphen

- flexibles Tool zur Modellierung
  - Transportnetzwerke



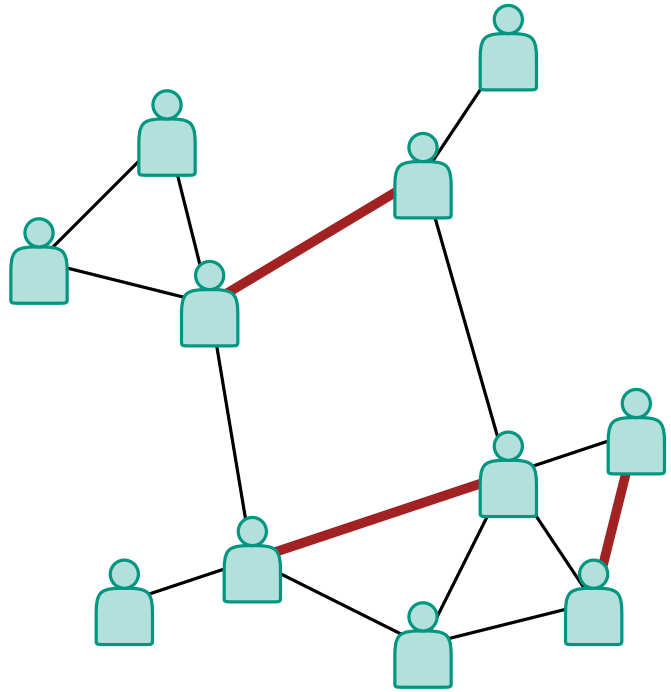
Graphen

- flexibles Tool zur Modellierung
  - Transportnetzwerke
  - soziale Netzwerke



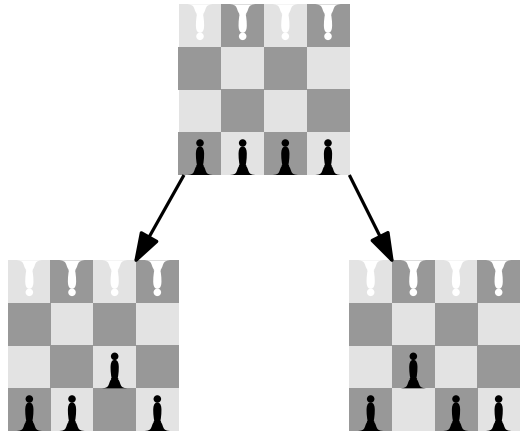
Graphen

- flexibles Tool zur Modellierung
  - Transportnetzwerke
  - soziale Netzwerke



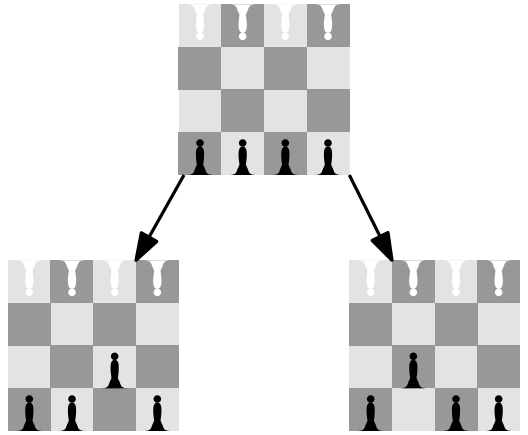
Graphen

- flexibles Tool zur Modellierung
  - Transportnetzwerke
  - soziale Netzwerke
  - sonstiges



Graphen

- flexibles Tool zur Modellierung
  - Transportnetzwerke
  - soziale Netzwerke
  - sonstiges
- viele praktische Fragestellungen sind algorithmische Probleme auf Graphen



Grundlagen und Notation

Grundlagen und Notation

Graph

$G = (V, E)$

Grundlagen und Notation

Graph

$G = (V, E)$

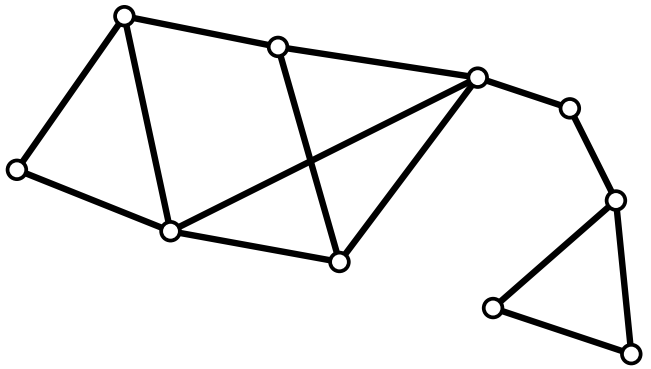
- V: endliche Menge
- E: Teilmenge von $\binom{V}{2}$

Grundlagen und Notation

Graph

$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $\binom{V}{2}$

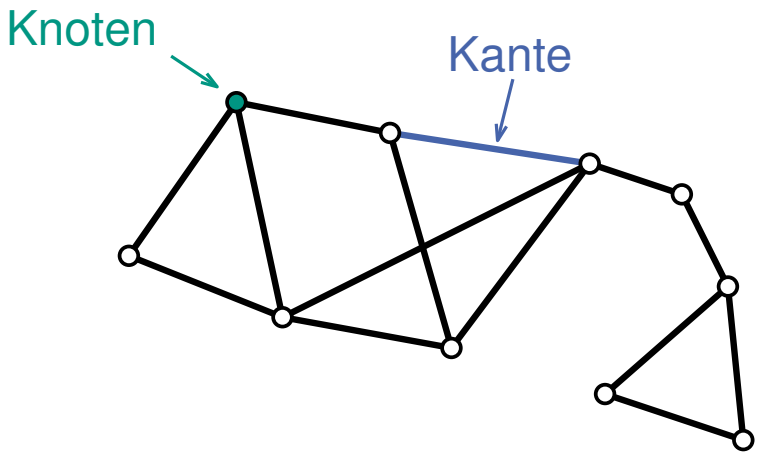


Grundlagen und Notation

Graph

$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $\binom{V}{2}$

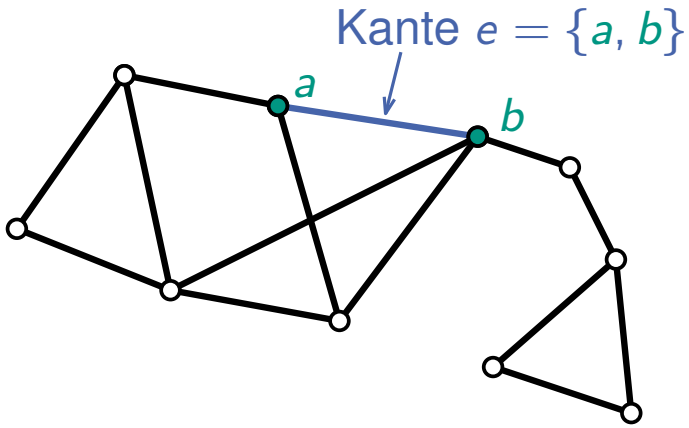


Grundlagen und Notation

Graph

$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $\binom{V}{2}$

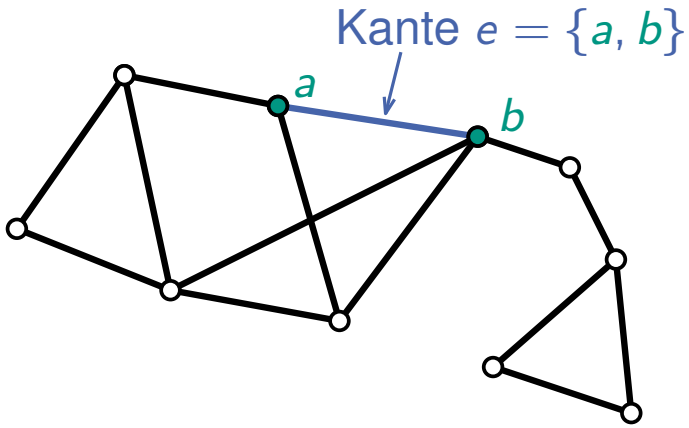


Grundlagen und Notation

Graph

$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $\binom{V}{2}$



Relationen

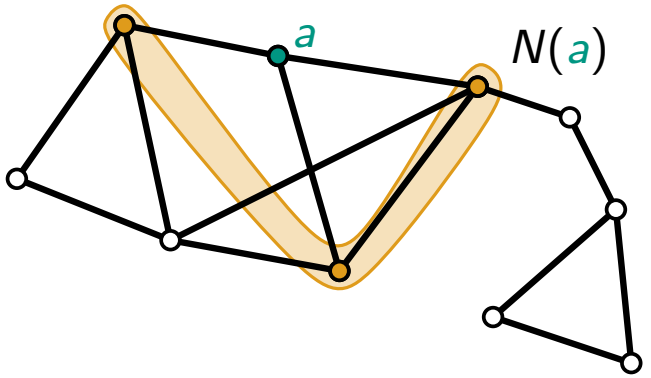
- a adjazent zu b
- e inzident zu a, b
- $b \in N(a)$

Grundlagen und Notation

Graph

$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $\binom{V}{2}$



Relationen

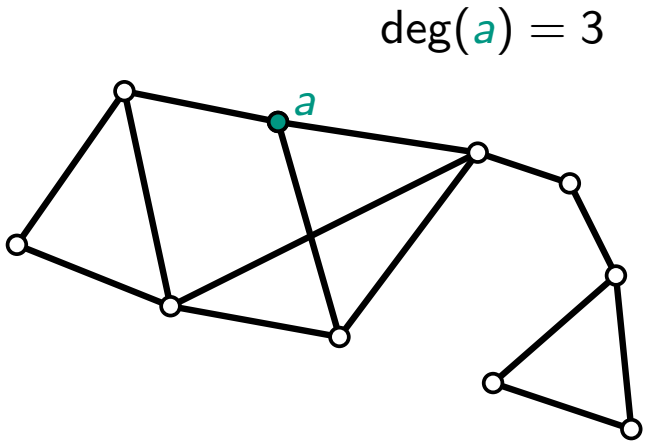
- a adjazent zu b
- e inzident zu a, b
- $b \in N(a)$

Grundlagen und Notation

Graph

$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $\binom{V}{2}$



Relationen

- a adjazent zu b
- e inzident zu a, b
- $b \in N(a)$

Grundlagen und Notation

Graph

$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $\binom{V}{2}$

Graph (gerichtet)

Grundlagen und Notation

Graph

$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $\binom{V}{2}$

Graph (gerichtet)

$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $V \times V$

Grundlagen und Notation

Graph

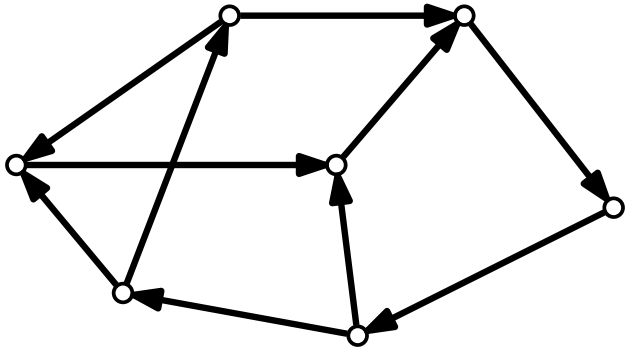
$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $\binom{V}{2}$

Graph (gerichtet)

$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $V \times V$



Grundlagen und Notation

Graph

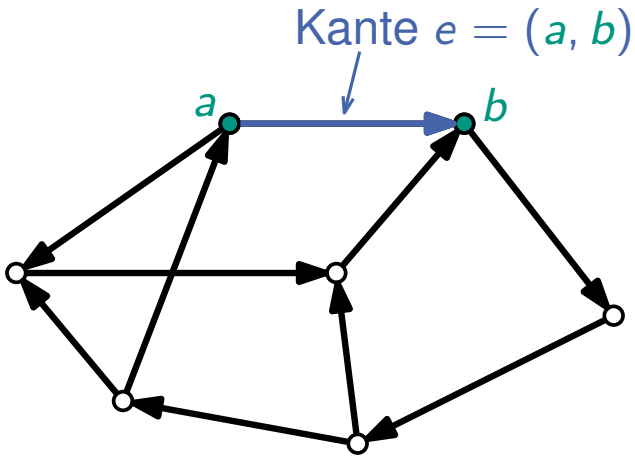
$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $\binom{V}{2}$

Graph (gerichtet)

$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $V \times V$



Grundlagen und Notation

Graph

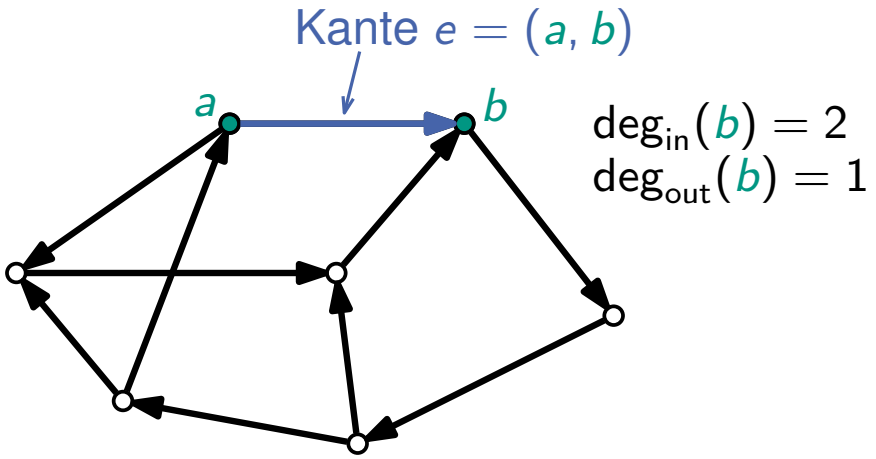
$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $\binom{V}{2}$

Graph (gerichtet)

$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $V \times V$



Grundlagen und Notation

Graph

$G = (V, E)$

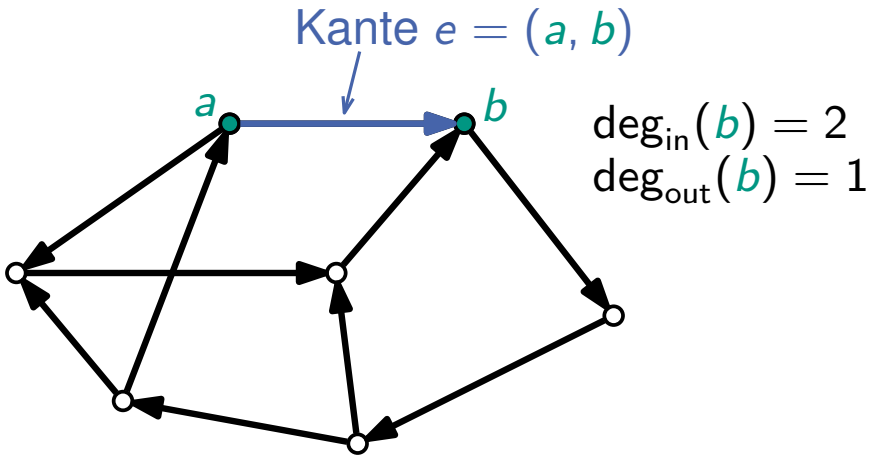
- V: endliche Menge
- E: Teilmenge von $\binom{V}{2}$

Graph (gerichtet)

$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $V \times V$

Sonstiges



Grundlagen und Notation

Graph

$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $\binom{V}{2}$

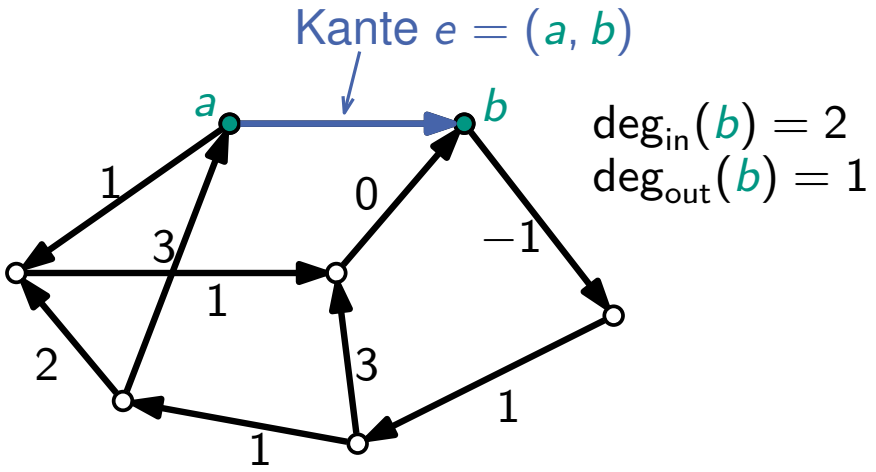
Graph (gerichtet)

$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $V \times V$

Sonstiges

- Gewichte: $G = (V, E, w)$ mit $w : E \mapsto \mathbb{R}$



Grundlagen und Notation

Graph

$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $\binom{V}{2}$

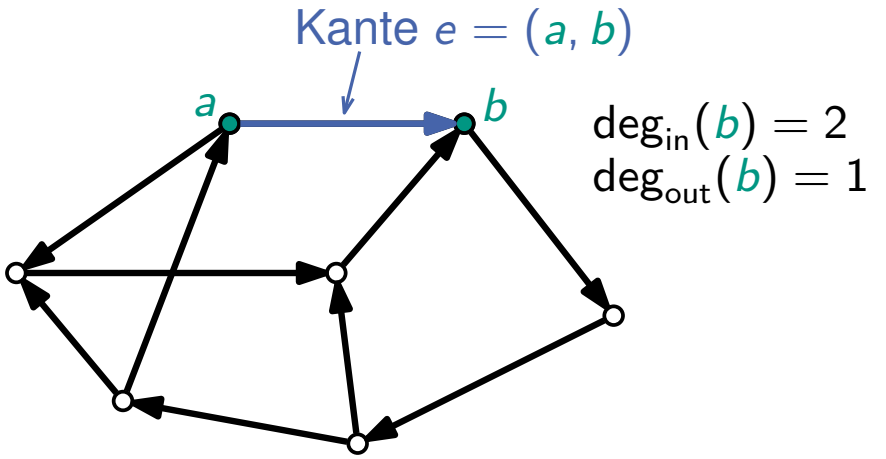
Graph (gerichtet)

$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $V \times V$

Sonstiges

- Gewichte: $G = (V, E, w)$ mit $w : E \mapsto \mathbb{R}$



Grundlagen und Notation

Graph

$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $\binom{V}{2}$

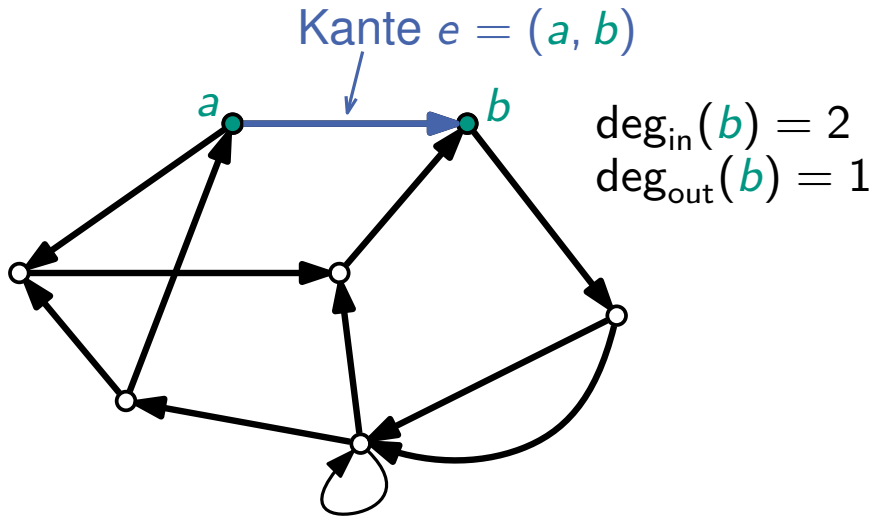
Graph (gerichtet)

$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $V \times V$

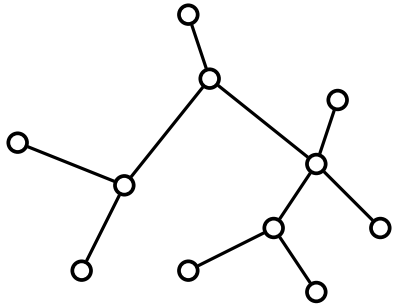
Sonstiges

- Gewichte: $G = (V, E, w)$ mit $w : E \mapsto \mathbb{R}$
- Multigraphen



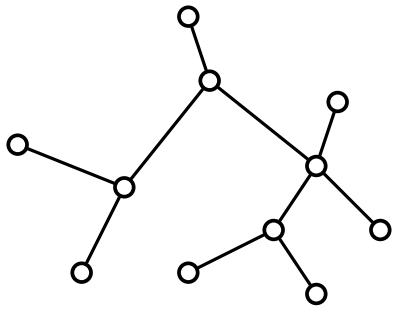
Besondere Graphen

Besondere Graphen



Bäume

Besondere Graphen

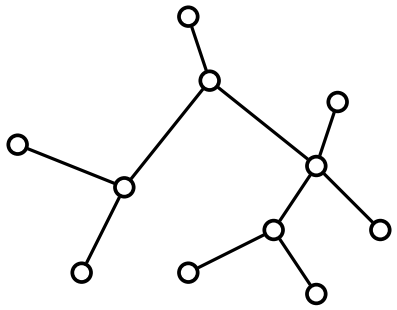


Bäume

Charakterisierung:

- kreisfrei
- zusammenhängend
- $m = n - 1$

Besondere Graphen



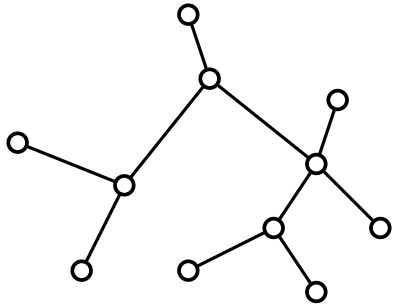
Bäume

Charakterisierung:

- kreisfrei
- zusammenhängend
- $m = n - 1$

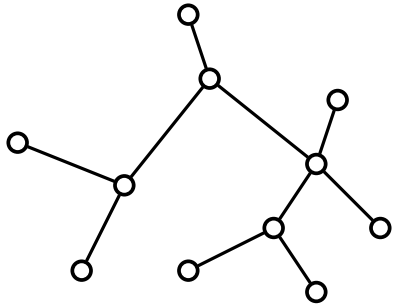
} Zwei Eigenschaften implizieren dritte

Besondere Graphen

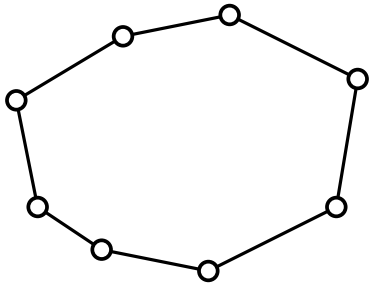


Bäume

Besondere Graphen

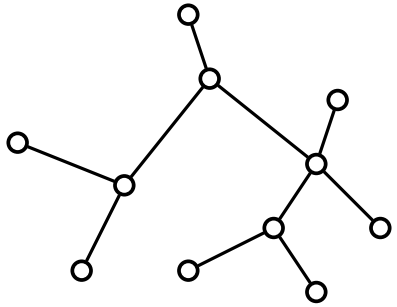


Bäume

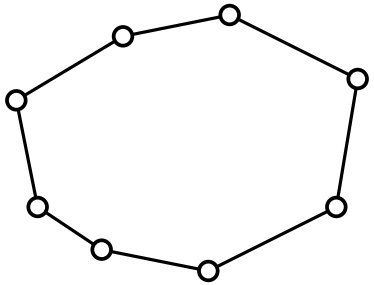


Kreise C_n

Besondere Graphen



Bäume

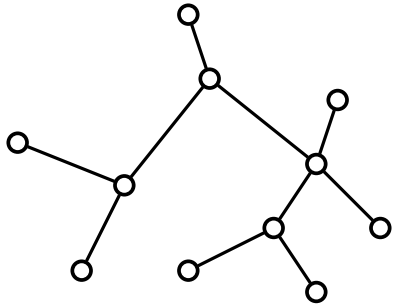


Kreise C_n

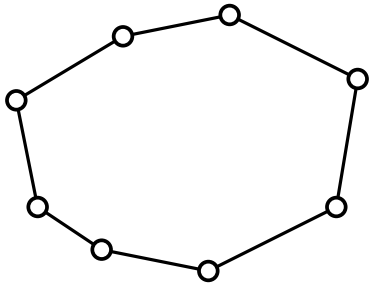
Charakterisierung:

- zusammenhängend
- $\deg(v) = 2$ f.a. $v \in V$

Besondere Graphen



Bäume



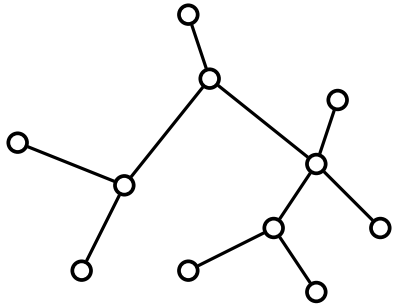
Kreise C_n

Charakterisierung:

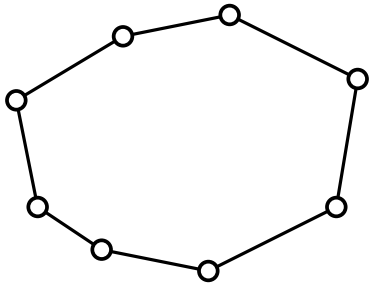
- zusammenhängend
- $\deg(v) = 2$ f.a. $v \in V$

Argumentation: $\forall v \in V : \deg(v) = 2 \Rightarrow G$ ist Kreis

Besondere Graphen



Bäume



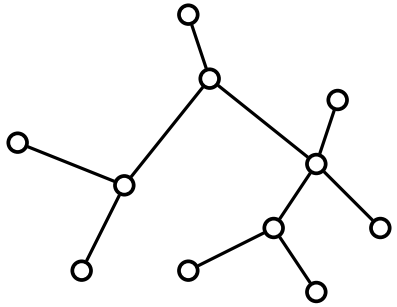
Kreise C_n

Charakterisierung:

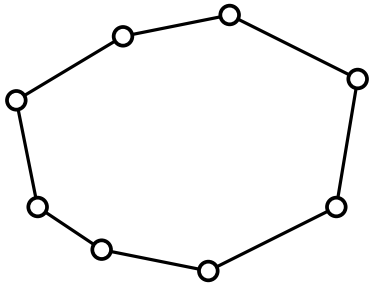
- zusammenhängend
- $\deg(v) = 2$ f.a. $v \in V$

Argumentation: $\forall v \in V : \deg(v) = 2 \Rightarrow G$ ist Kreis

Besondere Graphen



Bäume



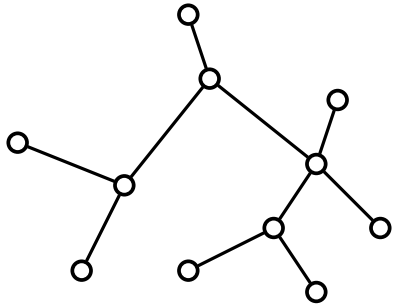
Kreise C_n

Charakterisierung:

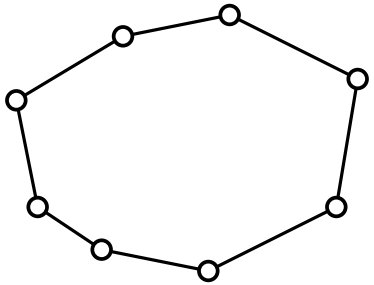
- zusammenhängend
- $\deg(v) = 2$ f.a. $v \in V$

Argumentation: $\forall v \in V : \deg(v) = 2 \Rightarrow G$ ist Kreis

Besondere Graphen



Bäume

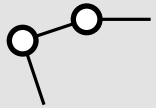


Kreise C_n

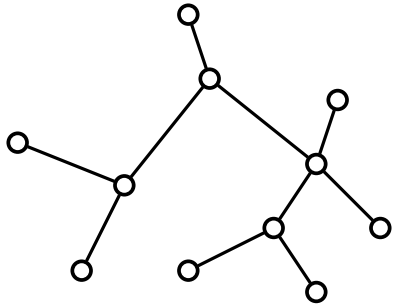
Charakterisierung:

- zusammenhängend
- $\deg(v) = 2$ f.a. $v \in V$

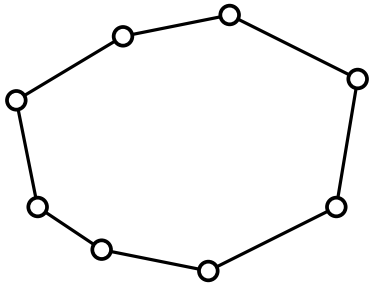
Argumentation: $\forall v \in V : \deg(v) = 2 \Rightarrow G$ ist Kreis



Besondere Graphen



Bäume

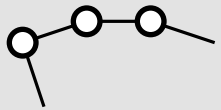


Kreise C_n

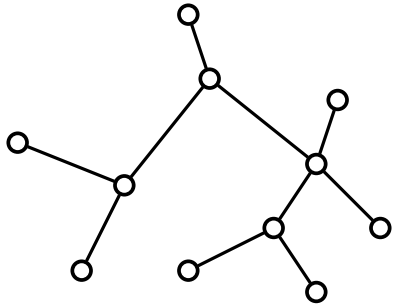
Charakterisierung:

- zusammenhängend
- $\deg(v) = 2$ f.a. $v \in V$

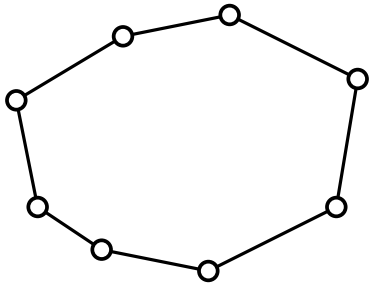
Argumentation: $\forall v \in V : \deg(v) = 2 \Rightarrow G$ ist Kreis



Besondere Graphen



Bäume

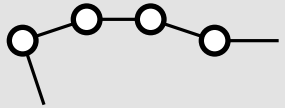


Kreise C_n

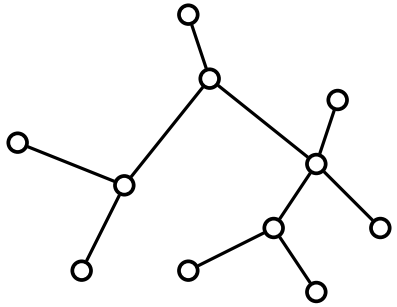
Charakterisierung:

- zusammenhängend
- $\deg(v) = 2$ f.a. $v \in V$

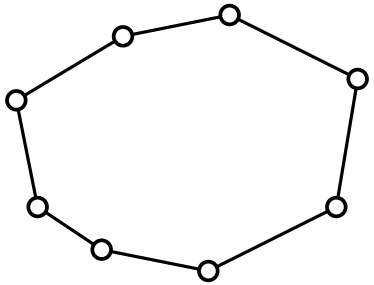
Argumentation: $\forall v \in V : \deg(v) = 2 \Rightarrow G$ ist Kreis



Besondere Graphen



Bäume

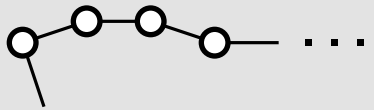


Kreise C_n

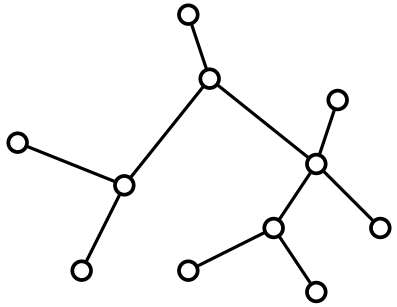
Charakterisierung:

- zusammenhängend
- $\deg(v) = 2$ f.a. $v \in V$

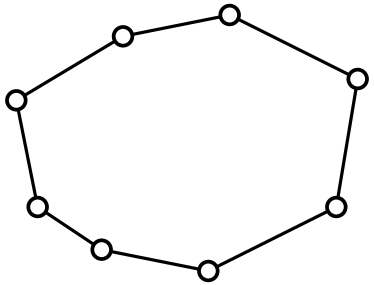
Argumentation: $\forall v \in V : \deg(v) = 2 \Rightarrow G$ ist Kreis



Besondere Graphen



Bäume

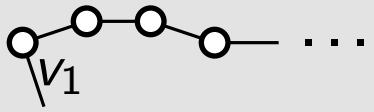


Kreise C_n

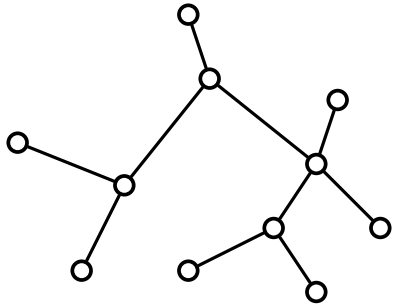
Charakterisierung:

- zusammenhängend
- $\deg(v) = 2$ f.a. $v \in V$

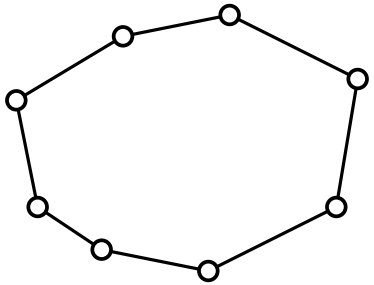
Argumentation: $\forall v \in V : \deg(v) = 2 \Rightarrow G$ ist Kreis



Besondere Graphen



Bäume

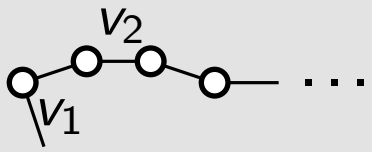


Kreise C_n

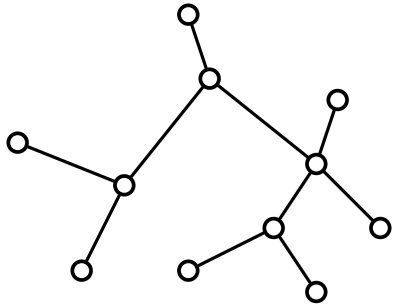
Charakterisierung:

- zusammenhängend
- $\deg(v) = 2$ f.a. $v \in V$

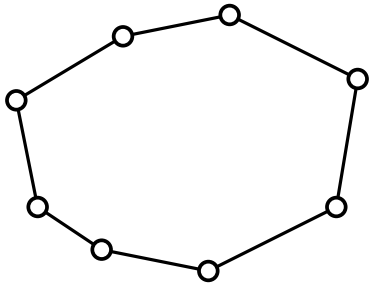
Argumentation: $\forall v \in V : \deg(v) = 2 \Rightarrow G$ ist Kreis



Besondere Graphen



Bäume

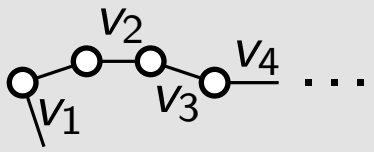


Kreise C_n

Charakterisierung:

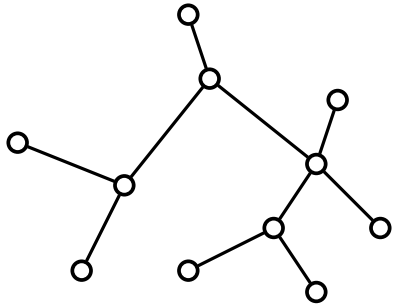
- zusammenhängend
- $\deg(v) = 2$ f.a. $v \in V$

Argumentation: $\forall v \in V : \deg(v) = 2 \Rightarrow G$ ist Kreis

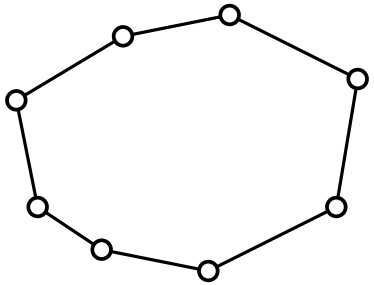


...

Besondere Graphen



Bäume

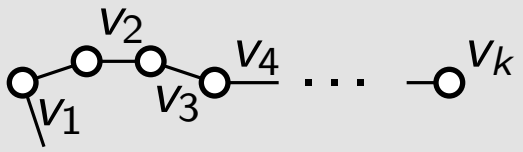


Kreise C_n

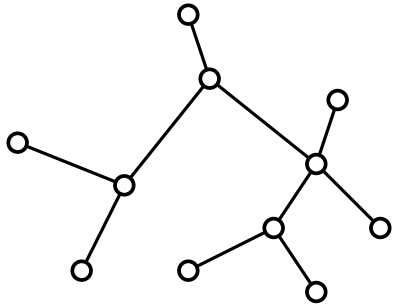
Charakterisierung:

- zusammenhängend
- $\deg(v) = 2$ f.a. $v \in V$

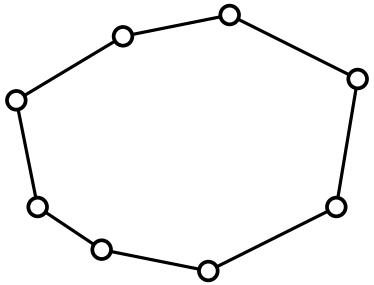
Argumentation: $\forall v \in V : \deg(v) = 2 \Rightarrow G$ ist Kreis



Besondere Graphen



Bäume

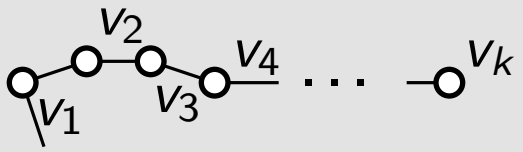


Kreise C_n

Charakterisierung:

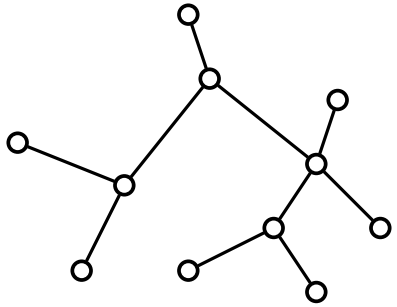
- zusammenhängend
- $\deg(v) = 2$ f.a. $v \in V$

Argumentation: $\forall v \in V : \deg(v) = 2 \Rightarrow G$ ist Kreis

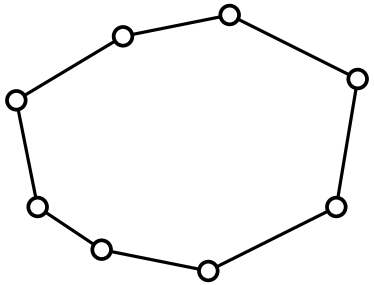


- für $2 \leq i < k$: $\{v_i, v_k\} \notin E$

Besondere Graphen



Bäume

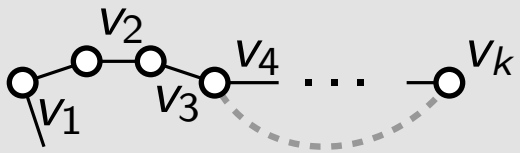


Kreise C_n

Charakterisierung:

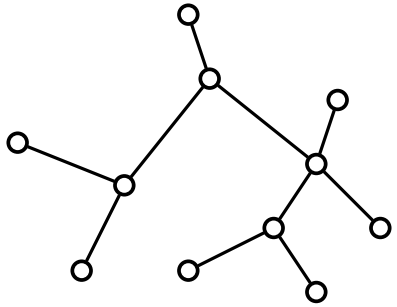
- zusammenhängend
- $\deg(v) = 2$ f.a. $v \in V$

Argumentation: $\forall v \in V : \deg(v) = 2 \Rightarrow G$ ist Kreis

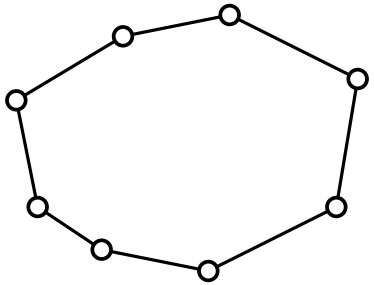


- für $2 \leq i < k$: $\{v_i, v_k\} \notin E$

Besondere Graphen



Bäume

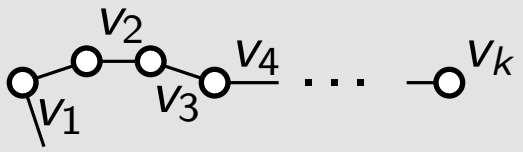


Kreise C_n

Charakterisierung:

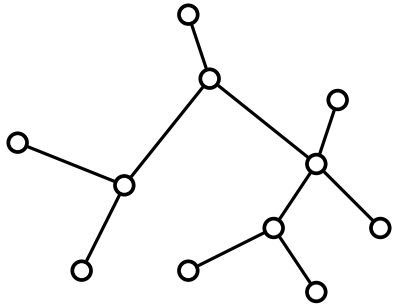
- zusammenhängend
- $\deg(v) = 2$ f.a. $v \in V$

Argumentation: $\forall v \in V : \deg(v) = 2 \Rightarrow G$ ist Kreis

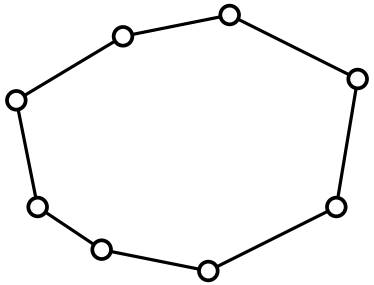


- für $2 \leq i < k$: $\{v_i, v_k\} \notin E$

Besondere Graphen



Bäume

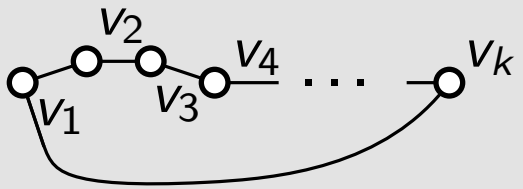


Kreise C_n

Charakterisierung:

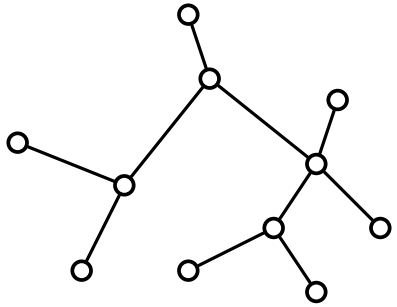
- zusammenhängend
- $\deg(v) = 2$ f.a. $v \in V$

Argumentation: $\forall v \in V : \deg(v) = 2 \Rightarrow G$ ist Kreis

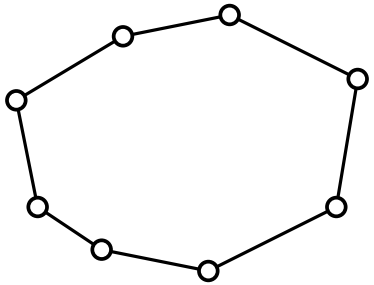


- für $2 \leq i < k$: $\{v_i, v_k\} \notin E$

Besondere Graphen



Bäume

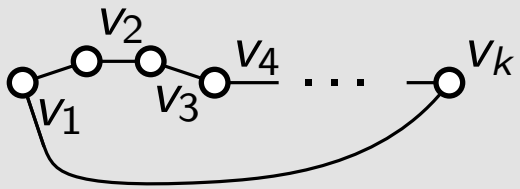


Kreise C_n

Charakterisierung:

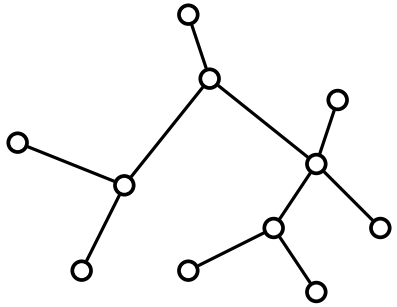
- zusammenhängend
- $\deg(v) = 2$ f.a. $v \in V$

Argumentation: $\forall v \in V : \deg(v) = 2 \Rightarrow G$ ist Kreis

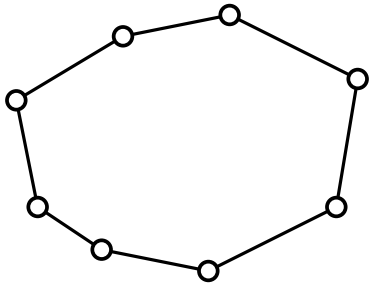


- für $2 \leq i < k$: $\{v_i, v_k\} \notin E$
- $\{v_k, v_1\} \in E$ falls $k = n$

Besondere Graphen

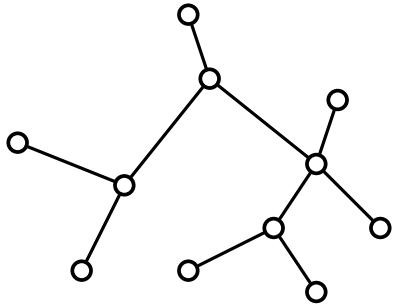


Bäume

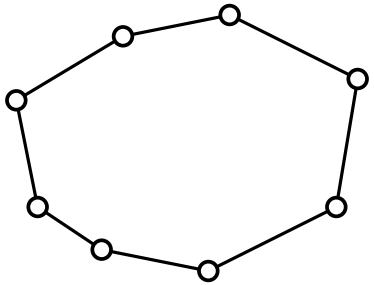


Kreise

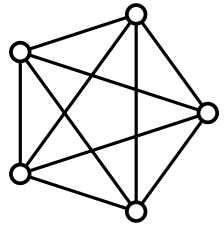
Besondere Graphen



Bäume

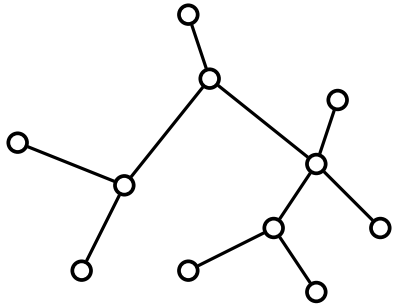


Kreise

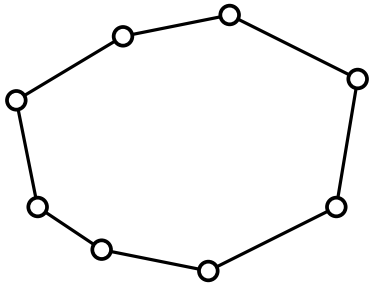


vollst. Graph K_n

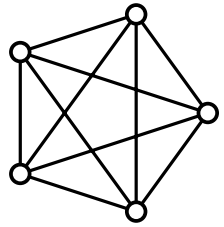
Besondere Graphen



Bäume



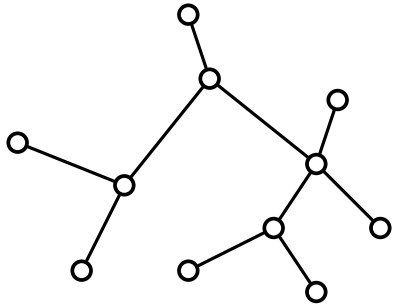
Kreise



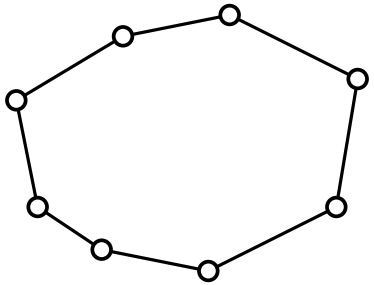
$m = \binom{n}{2} = \frac{n(n-1)}{2}$

vollst. Graph K_n

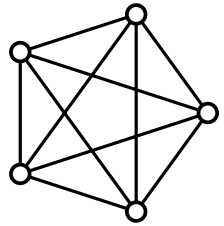
Besondere Graphen



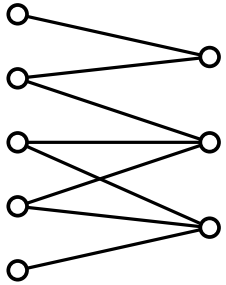
Bäume



Kreise

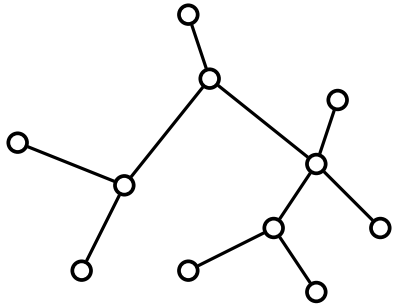


vollst. Graph K_n

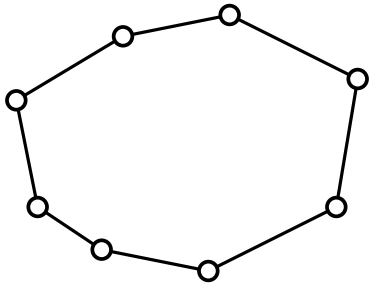


bipartiter Graph

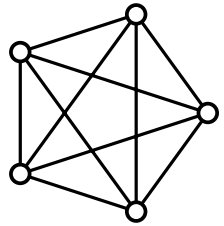
Besondere Graphen



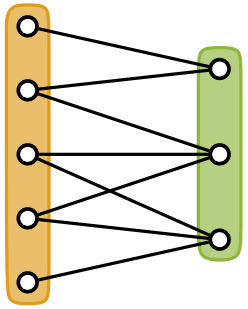
Bäume



Kreise



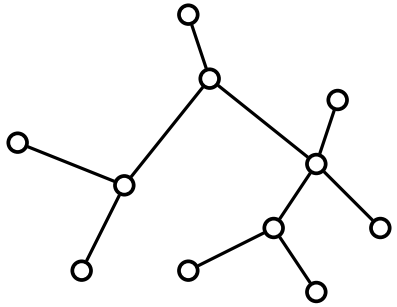
vollst. Graph K_n



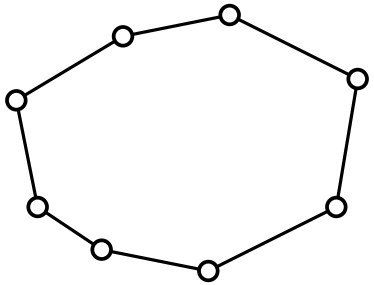
bipartiter Graph

$G = (A \cup B, E)$

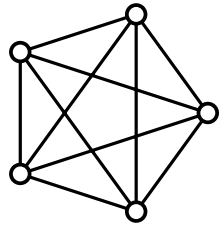
Besondere Graphen



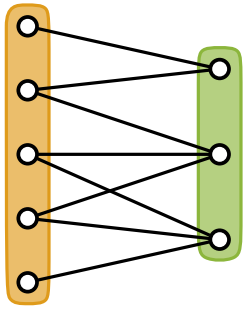
Bäume



Kreise



vollst. Graph K_n

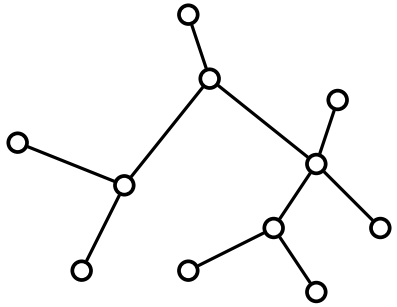


bipartiter Graph

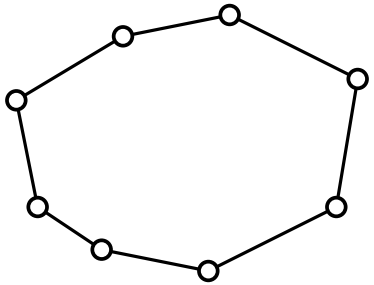
$G = (A \cup B, E)$

Charakterisierung:
G enthält nur Kreise *gerader* Länge in G

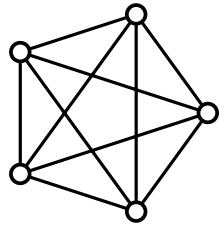
Besondere Graphen



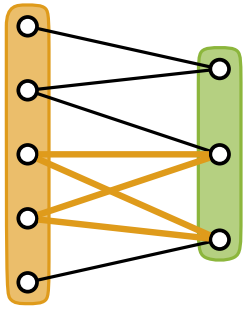
Bäume



Kreise



vollst. Graph K_n

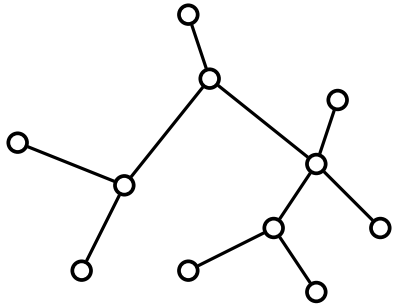


bipartiter Graph

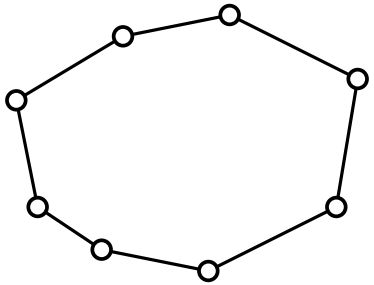
$G = (A \cup B, E)$

Charakterisierung:
G enthält nur Kreise *gerader* Länge in G

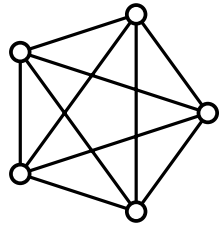
Besondere Graphen



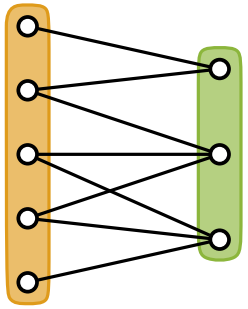
Bäume



Kreise



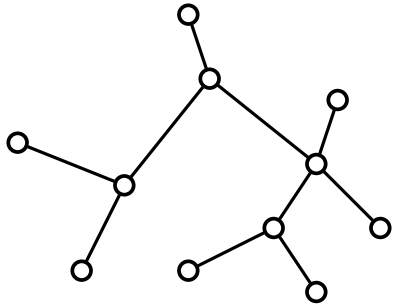
vollst. Graph K_n



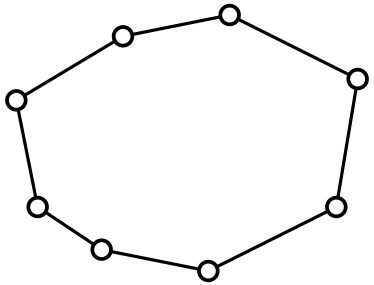
bipartiter Graph

$G = (A \cup B, E)$

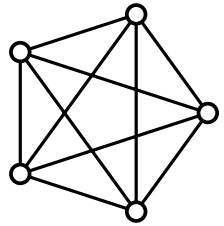
Besondere Graphen



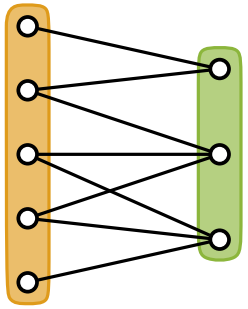
Bäume



Kreise

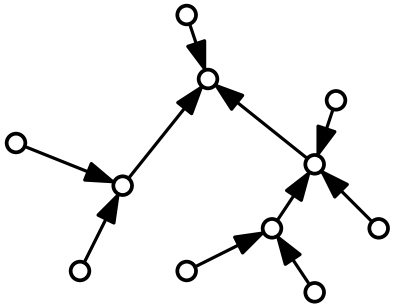


vollst. Graph K_n

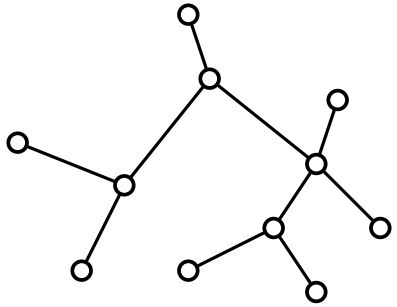


bipartiter Graph

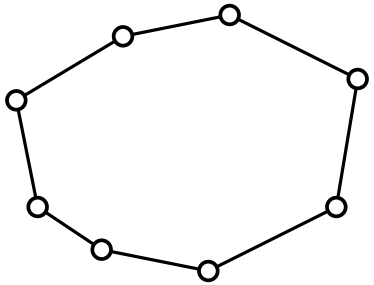
$G = (A \cup B, E)$



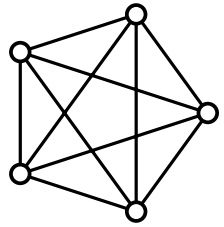
Besondere Graphen



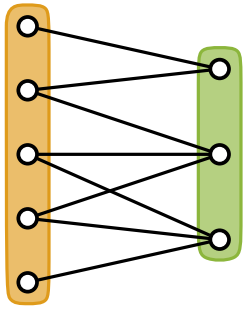
Bäume



Kreise

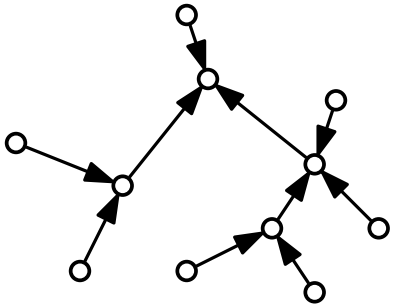


vollst. Graph K_n



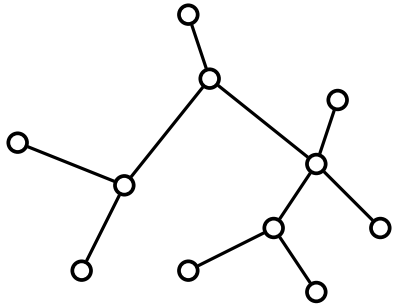
bipartiter Graph

$G = (A \cup B, E)$

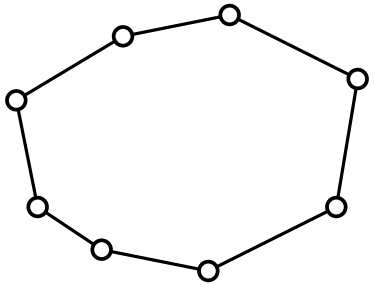


In-tree / zur Wurzel ger. Baum

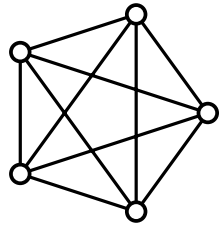
Besondere Graphen



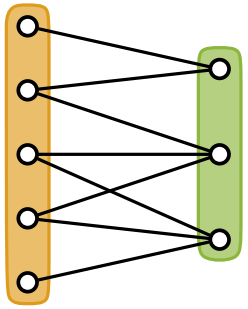
Bäume



Kreise

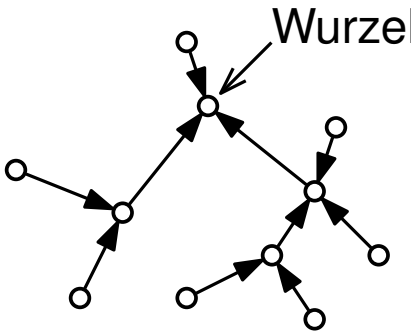


vollst. Graph K_n



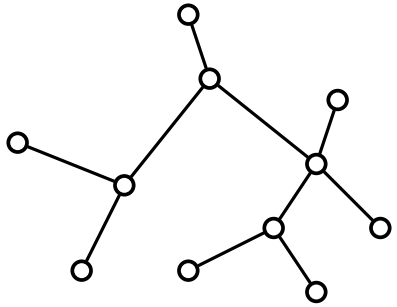
bipartiter Graph

$G = (A \cup B, E)$

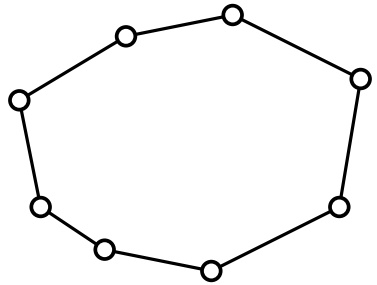


In-tree / zur Wurzel ger. Baum

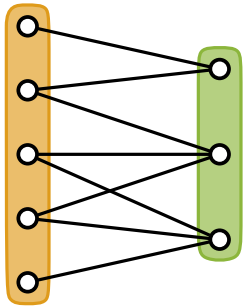
Besondere Graphen



Bäume

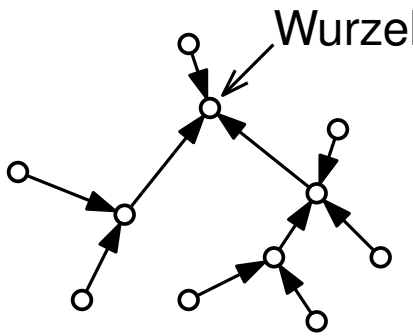


Kreise



bipartiter Graph

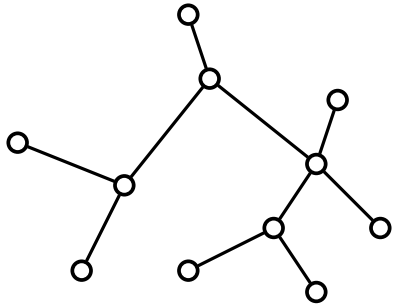
$G = (A \cup B, E)$



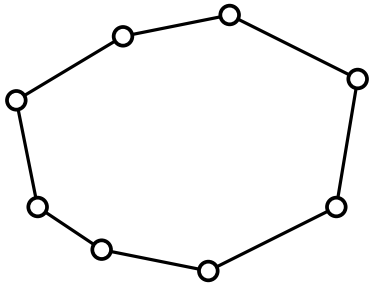
In-tree / zur Wurzel ger. Baum

Charakterisierung über $\deg_{out}$

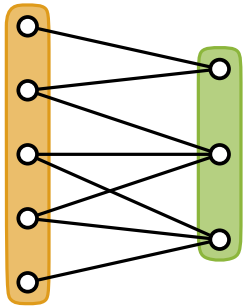
Besondere Graphen



Bäume

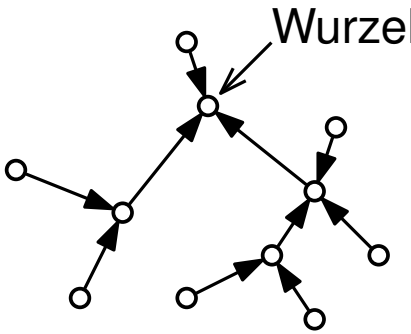


Kreise



bipartiter Graph

$G = (A \cup B, E)$

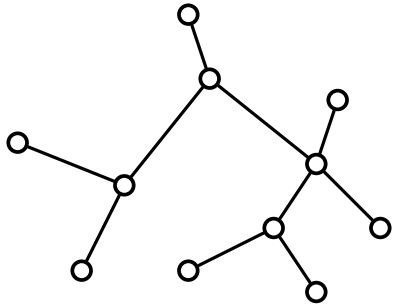


In-tree / zur Wurzel ger. Baum

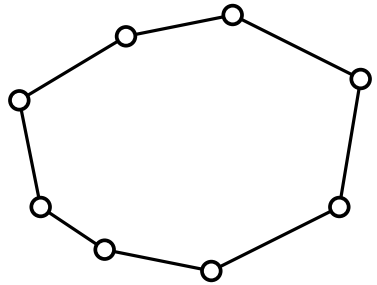
Charakterisierung über $\deg_{\text{out}}$

Ein Knoten r mit $\deg_{\text{out}}(r) = 0$

Besondere Graphen



Bäume



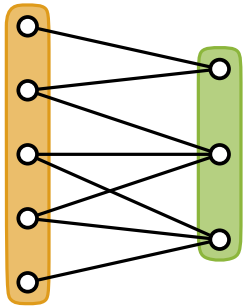
Kreise

Charakterisierung über $\deg_{out}$

Ein Knoten r mit $\deg_{out}(r) = 0$

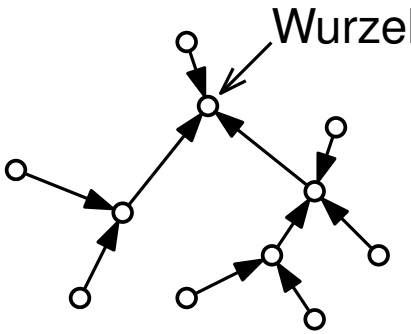
Jeder andere Knoten v:

$$\deg_{out}(v) = 1$$



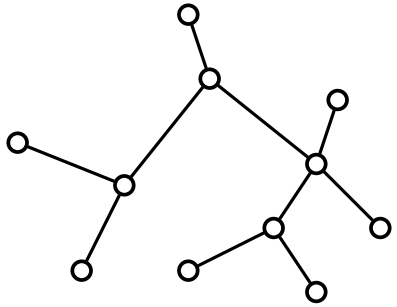
bipartiter Graph

$G = (A \cup B, E)$

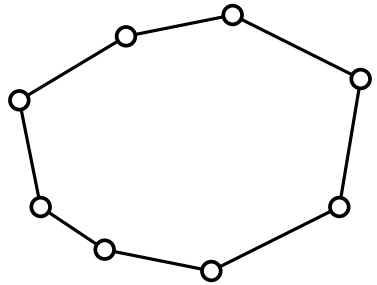


In-tree / zur Wurzel ger. Baum

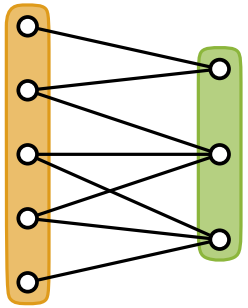
Besondere Graphen



Bäume

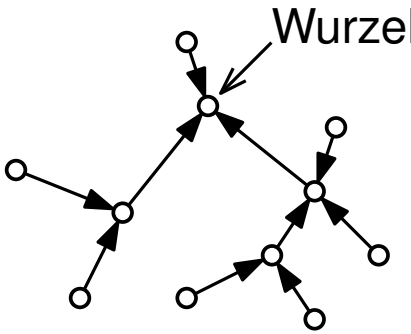


Kreise



bipartiter Graph

$G = (A \cup B, E)$



In-tree / zur Wurzel ger. Baum

Charakterisierung über $\deg_{\text{out}}$

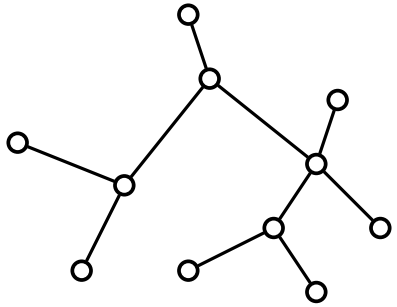
Ein Knoten r mit $\deg_{\text{out}}(r) = 0$

Jeder andere Knoten v:

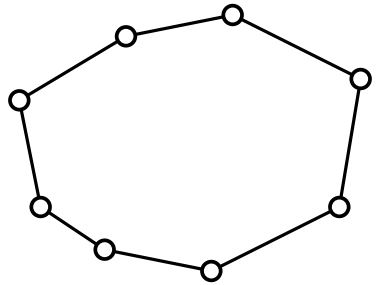
$$\deg_{\text{out}}(v) = 1$$

Beweis: $\Rightarrow$

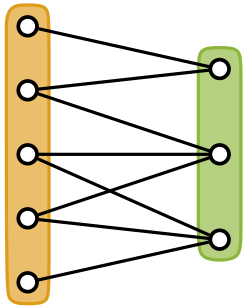
Besondere Graphen



Bäume

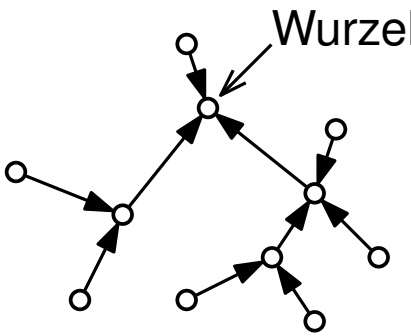


Kreise



bipartiter Graph

$G = (A \cup B, E)$



In-tree / zur Wurzel ger. Baum

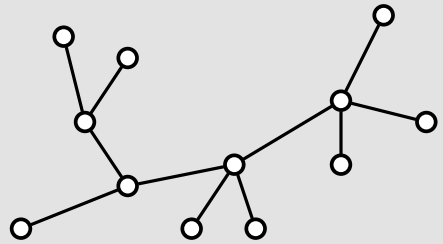
Charakterisierung über $\deg_{\text{out}}$

Ein Knoten r mit $\deg_{\text{out}}(r) = 0$

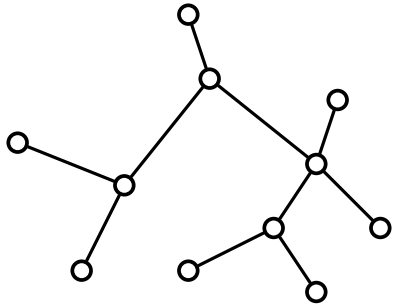
Jeder andere Knoten v:

$$\deg_{\text{out}}(v) = 1$$

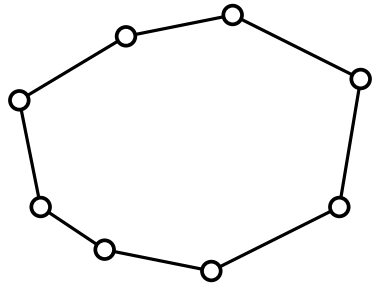
Beweis: $\Rightarrow$



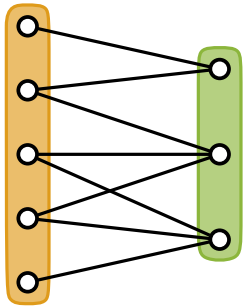
Besondere Graphen



Bäume

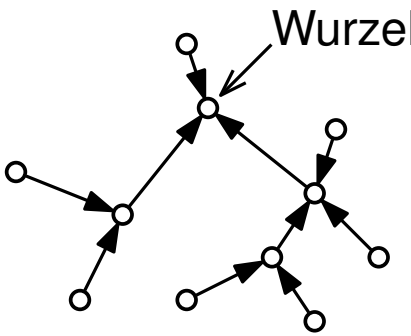


Kreise



bipartiter Graph

$G = (A \cup B, E)$



In-tree / zur Wurzel ger. Baum

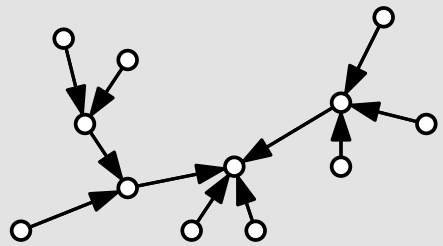
Charakterisierung über $\deg_{\text{out}}$

Ein Knoten r mit $\deg_{\text{out}}(r) = 0$

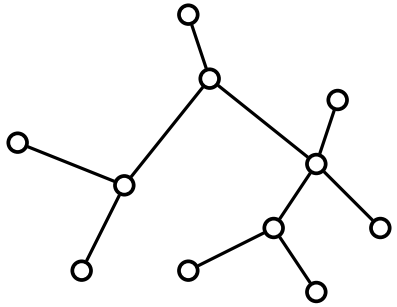
Jeder andere Knoten v:

$$\deg_{\text{out}}(v) = 1$$

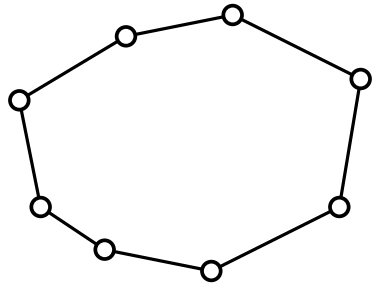
Beweis: $\Rightarrow$



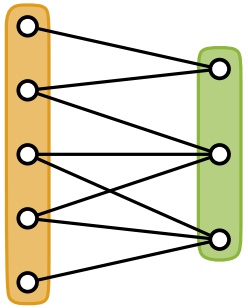
Besondere Graphen



Bäume

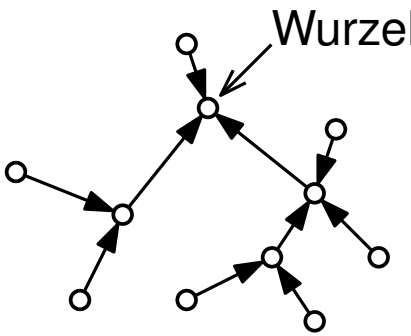


Kreise



bipartiter Graph

$G = (A \cup B, E)$



In-tree / zur Wurzel ger. Baum

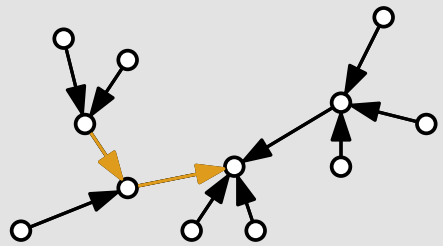
Charakterisierung über $\deg_{\text{out}}$

Ein Knoten r mit $\deg_{\text{out}}(r) = 0$

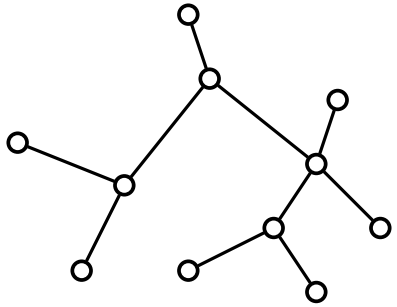
Jeder andere Knoten v:

$$\deg_{\text{out}}(v) = 1$$

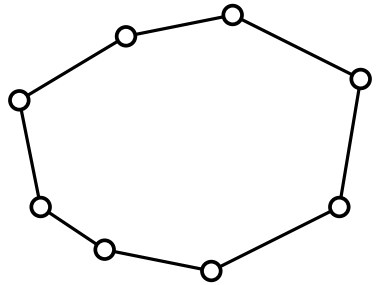
Beweis: $\Rightarrow$



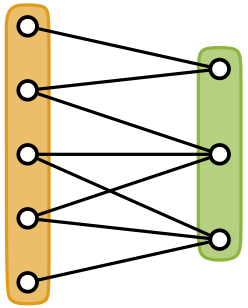
Besondere Graphen



Bäume

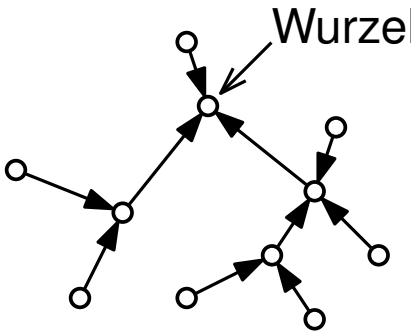


Kreise



bipartiter Graph

$G = (A \cup B, E)$



In-tree / zur Wurzel ger. Baum

Charakterisierung über $\deg_{\text{out}}$

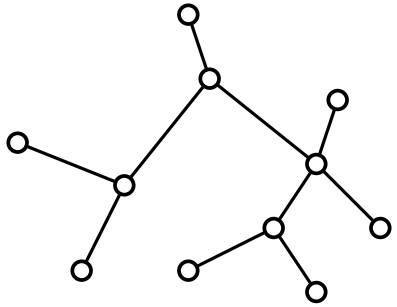
Ein Knoten r mit $\deg_{\text{out}}(r) = 0$

Jeder andere Knoten v:

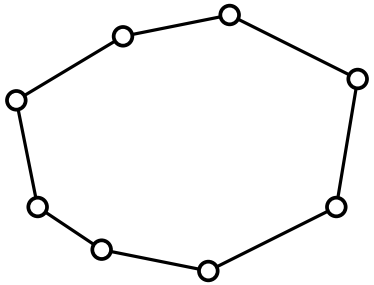
$$\deg_{\text{out}}(v) = 1$$

Beweis: $\Leftarrow$

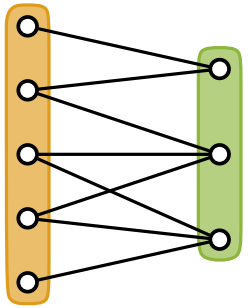
Besondere Graphen



Bäume

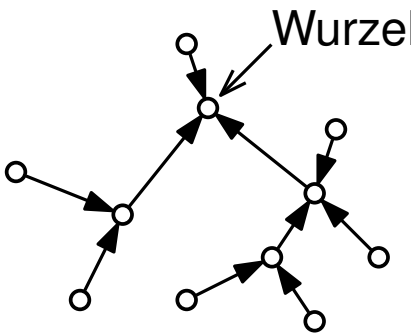


Kreise



bipartiter Graph

$G = (A \cup B, E)$



In-tree / zur Wurzel ger. Baum

Charakterisierung über $\deg_{\text{out}}$

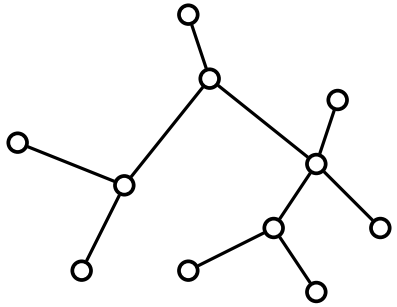
Ein Knoten r mit $\deg_{\text{out}}(r) = 0$

Jeder andere Knoten v:

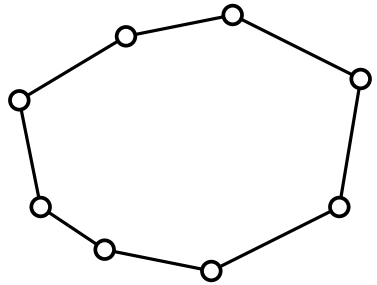
$$\deg_{\text{out}}(v) = 1$$

Beweis: $\Leftarrow$ per Induktion

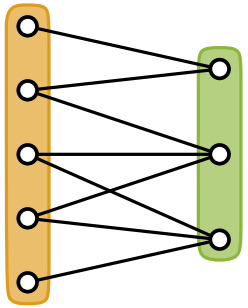
Besondere Graphen



Bäume

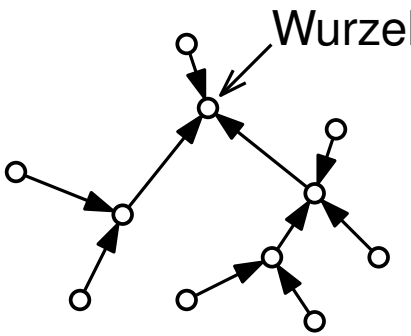


Kreise



bipartiter Graph

$G = (A \cup B, E)$



In-tree / zur Wurzel ger. Baum

Charakterisierung über $\deg_{\text{out}}$

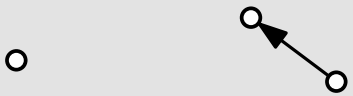
Ein Knoten r mit $\deg_{\text{out}}(r) = 0$

Jeder andere Knoten v:

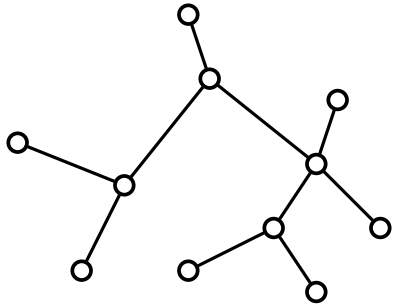
$$\deg_{\text{out}}(v) = 1$$

Beweis: $\Leftarrow$ per Induktion

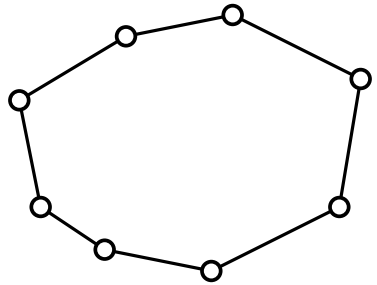
Anfang:



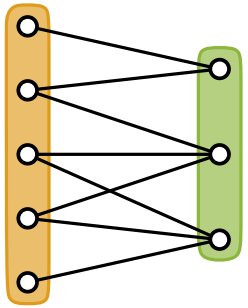
Besondere Graphen



Bäume

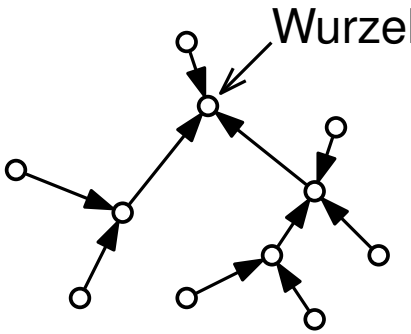


Kreise



bipartiter Graph

$G = (A \cup B, E)$



In-tree / zur Wurzel ger. Baum

Charakterisierung über $\deg_{\text{out}}$

Ein Knoten r mit $\deg_{\text{out}}(r) = 0$

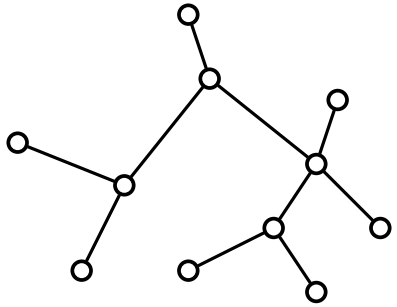
Jeder andere Knoten v:

$$\deg_{\text{out}}(v) = 1$$

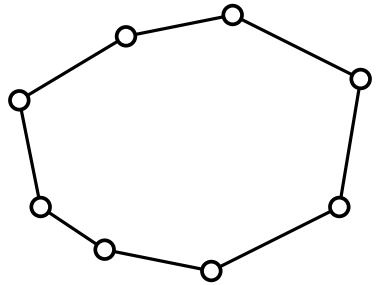
Beweis: $\Leftarrow$ per Induktion

Schritt:

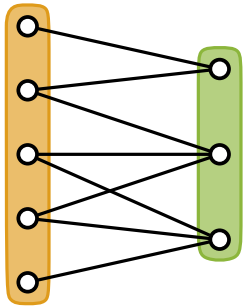
Besondere Graphen



Bäume



Kreise



bipartiter Graph

$G = (A \cup B, E)$

Charakterisierung über $\deg_{\text{out}}$

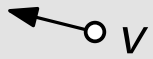
Ein Knoten r mit $\deg_{\text{out}}(r) = 0$

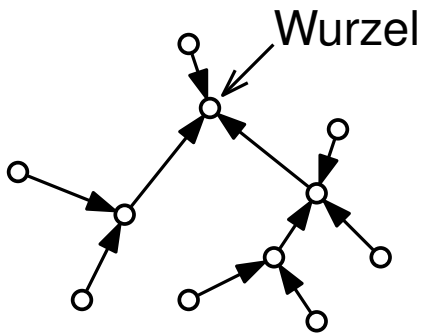
Jeder andere Knoten v:

$$\deg_{\text{out}}(v) = 1$$

Beweis: $\Leftarrow$ per Induktion

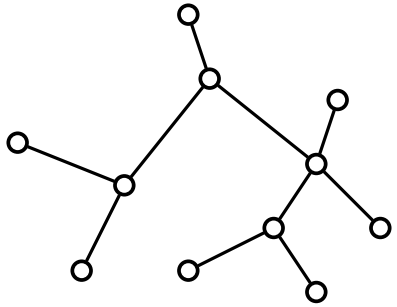
Schritt: $\deg_{\text{out}}(v) = 1$



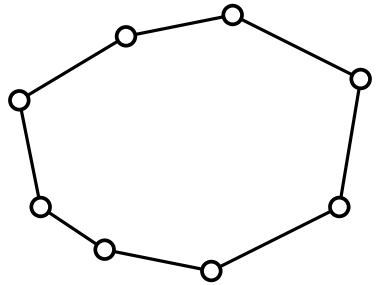


In-tree / zur Wurzel ger. Baum

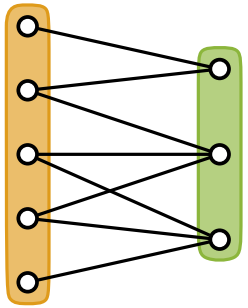
Besondere Graphen



Bäume



Kreise



bipartiter Graph

$G = (A \cup B, E)$

Charakterisierung über $\deg_{\text{out}}$

Ein Knoten r mit $\deg_{\text{out}}(r) = 0$

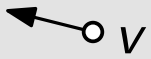
Jeder andere Knoten v:

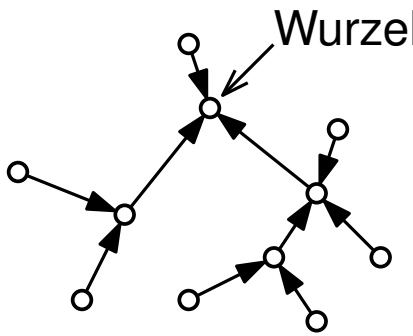
$$\deg_{\text{out}}(v) = 1$$

Beweis: $\Leftarrow$ per Induktion

Schritt: $\deg_{\text{out}}(v) = 1$

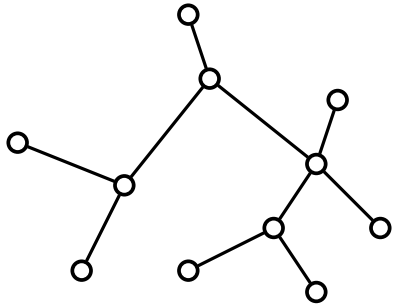
$\deg_{\text{in}}(v) = 0$



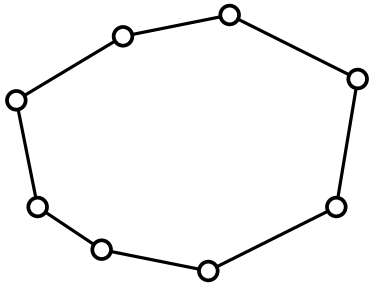


In-tree / zur Wurzel ger. Baum

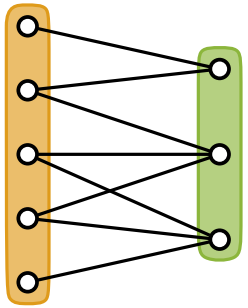
Besondere Graphen



Bäume

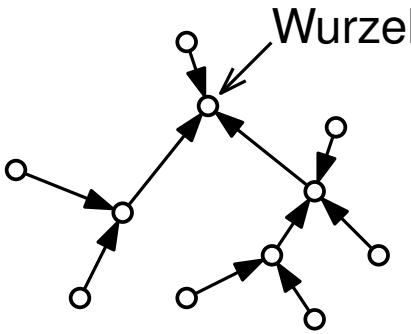


Kreise



bipartiter Graph

$G = (A \cup B, E)$



In-tree / zur Wurzel ger. Baum

Charakterisierung über $\deg_{\text{out}}$

Ein Knoten r mit $\deg_{\text{out}}(r) = 0$

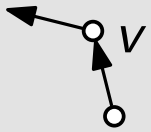
Jeder andere Knoten v:

$$\deg_{\text{out}}(v) = 1$$

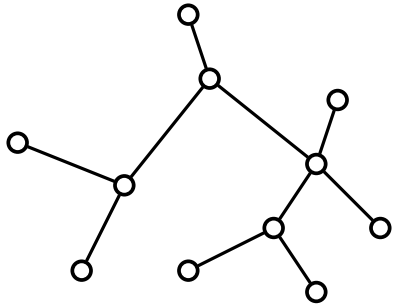
Beweis: $\Leftarrow$ per Induktion

Schritt: $\deg_{\text{out}}(v) = 1$

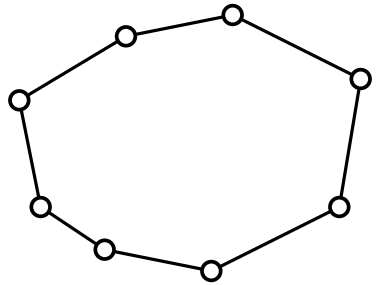
$\deg_{\text{in}}(v) = 0$



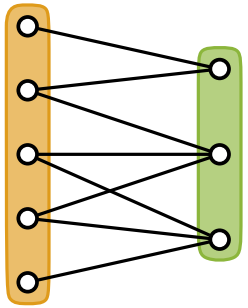
Besondere Graphen



Bäume

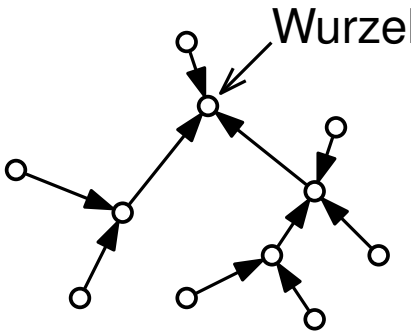


Kreise



bipartiter Graph

$G = (A \cup B, E)$



In-tree / zur Wurzel ger. Baum

Charakterisierung über $\deg_{\text{out}}$

Ein Knoten r mit $\deg_{\text{out}}(r) = 0$

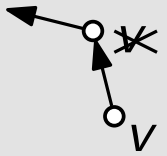
Jeder andere Knoten v:

$$\deg_{\text{out}}(v) = 1$$

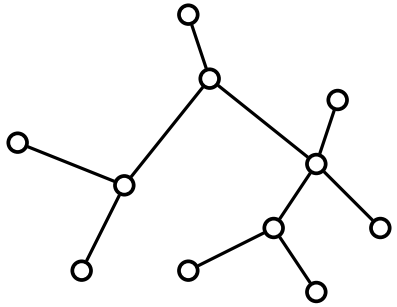
Beweis: $\Leftarrow$ per Induktion

Schritt: $\deg_{\text{out}}(v) = 1$

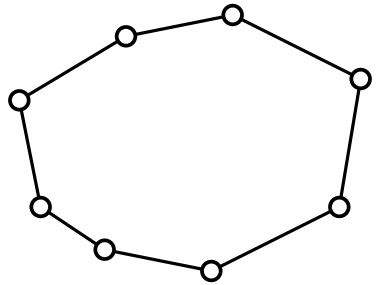
$\deg_{\text{in}}(v) = 0$



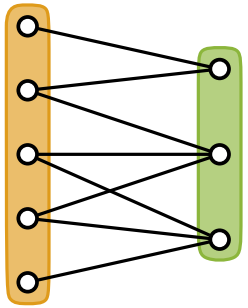
Besondere Graphen



Bäume

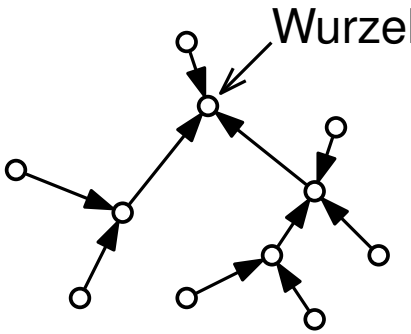


Kreise



bipartiter Graph

$G = (A \cup B, E)$



In-tree / zur Wurzel ger. Baum

Charakterisierung über $\deg_{\text{out}}$

Ein Knoten r mit $\deg_{\text{out}}(r) = 0$

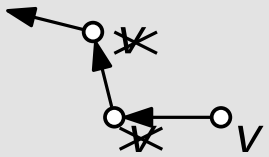
Jeder andere Knoten v:

$$\deg_{\text{out}}(v) = 1$$

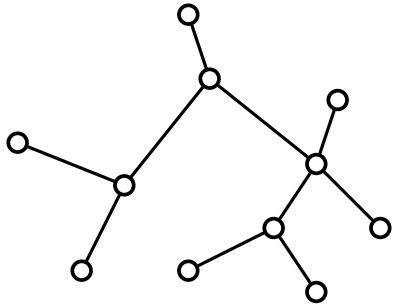
Beweis: $\Leftarrow$ per Induktion

Schritt: $\deg_{\text{out}}(v) = 1$

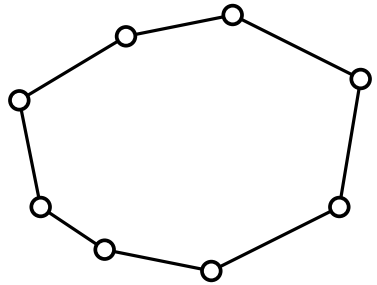
$\deg_{\text{in}}(v) = 0$



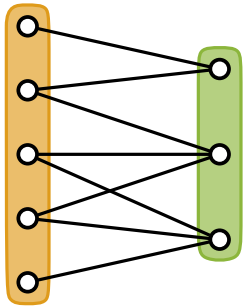
Besondere Graphen



Bäume

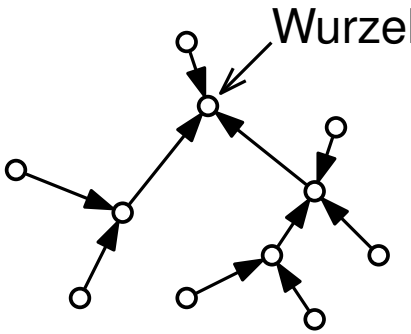


Kreise



bipartiter Graph

$G = (A \cup B, E)$



In-tree / zur Wurzel ger. Baum

Charakterisierung über $\deg_{\text{out}}$

Ein Knoten r mit $\deg_{\text{out}}(r) = 0$

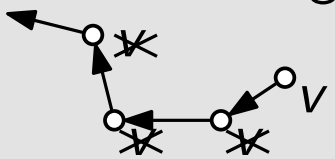
Jeder andere Knoten v:

$$\deg_{\text{out}}(v) = 1$$

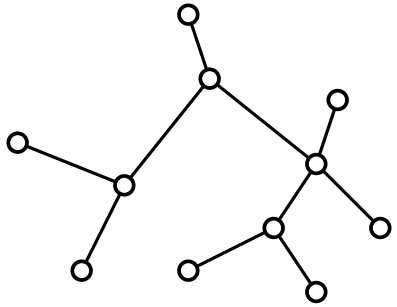
Beweis: $\Leftarrow$ per Induktion

Schritt: $\deg_{\text{out}}(v) = 1$

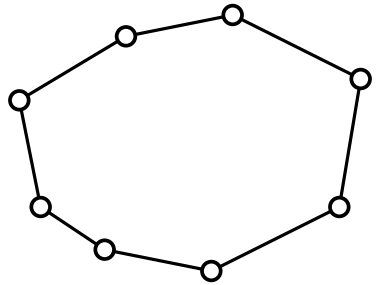
$\deg_{\text{in}}(v) = 0$



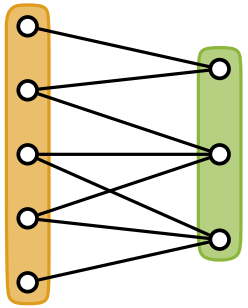
Besondere Graphen



Bäume

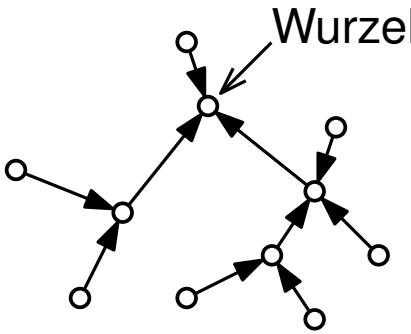


Kreise



bipartiter Graph

$G = (A \cup B, E)$



In-tree / zur Wurzel ger. Baum

Charakterisierung über $\deg_{out}$

Ein Knoten r mit $\deg_{out}(r) = 0$

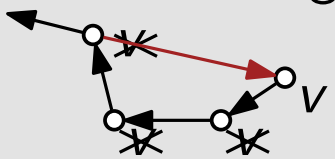
Jeder andere Knoten v:

$$\deg_{out}(v) = 1$$

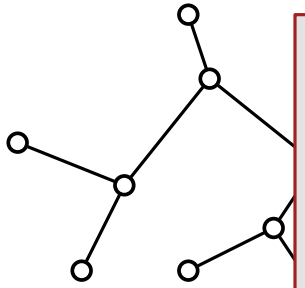
Beweis: $\Leftarrow$ per Induktion

Schritt: $\deg_{out}(v) = 1$

$\deg_{in}(v) = 0$



Besondere Graphen



Bäume

Achtung! Zusätzliche Forderung notwendig, damit Vorgänger von v keiner der bereits betrachteten Knoten sein kann:

- entweder kreisfreiheit
- oder: jeder Knoten hat Pfad zur Wurzel

Charakterisierung über $\deg_{\text{out}}$

Ein Knoten r mit $\deg_{\text{out}}(r) = 0$

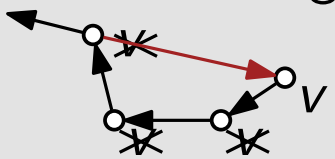
Jeder andere Knoten v:

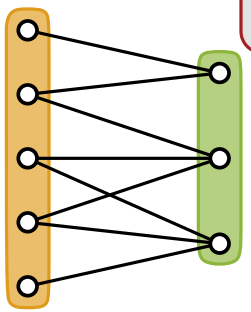
$$\deg_{\text{out}}(v) = 1$$

Beweis: $\Leftarrow$ per Induktion

Schritt: $\deg_{\text{out}}(v) = 1$

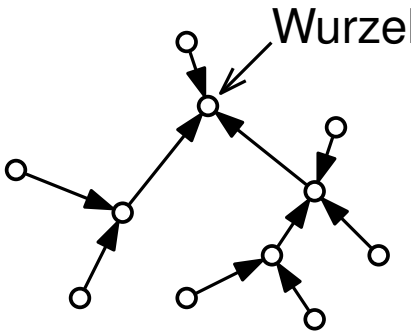
$\deg_{\text{in}}(v) = 0$





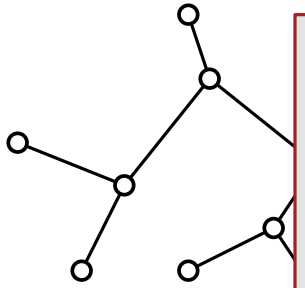
bipartiter Graph

$G = (A \cup B, E)$



In-tree / zur Wurzel ger. Baum

Besondere Graphen



Bäume

Achtung! Zusätzliche Forderung notwendig, damit Vorgänger von v keiner der bereits betrachteten Knoten sein kann:

- entweder kreisfreiheit
- oder: jeder Knoten hat Pfad zur Wurzel

Charakterisierung über $\deg_{\text{out}}$

Ein Knoten r mit $\deg_{\text{out}}(r) = 0$

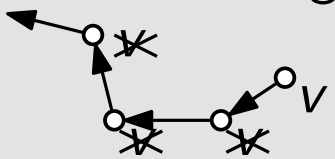
Jeder andere Knoten v:

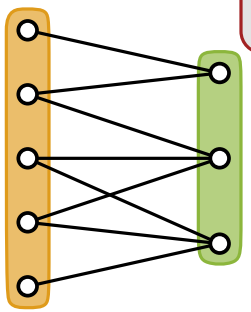
$$\deg_{\text{out}}(v) = 1$$

Beweis: $\Leftarrow$ per Induktion

Schritt: $\deg_{\text{out}}(v) = 1$

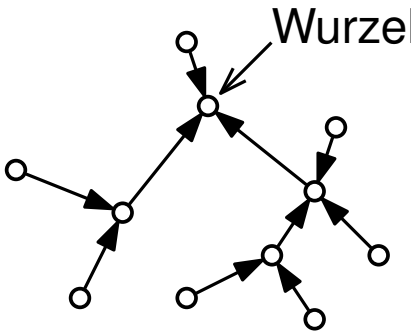
$\deg_{\text{in}}(v) = 0$





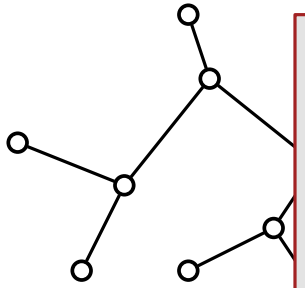
bipartiter Graph

$G = (A \cup B, E)$



In-tree / zur Wurzel ger. Baum

Besondere Graphen



Bäume

Achtung! Zusätzliche Forderung notwendig, damit Vorgänger von v keiner der bereits betrachteten Knoten sein kann:

- entweder kreisfreiheit
- oder: jeder Knoten hat Pfad zur Wurzel

Charakterisierung über $\deg_{\text{out}}$

Ein Knoten r mit $\deg_{\text{out}}(r) = 0$

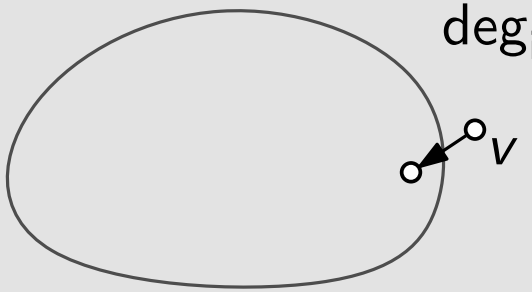
Jeder andere Knoten v:

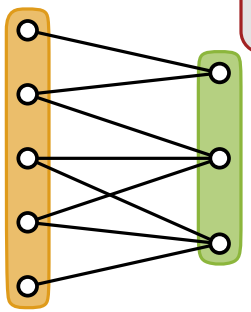
$$\deg_{\text{out}}(v) = 1$$

Beweis: $\Leftarrow$ per Induktion

Schritt: $\deg_{\text{out}}(v) = 1$

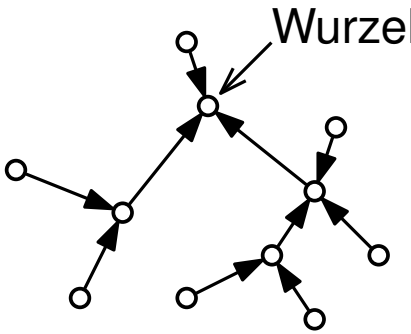
$\deg_{\text{in}}(v) = 0$





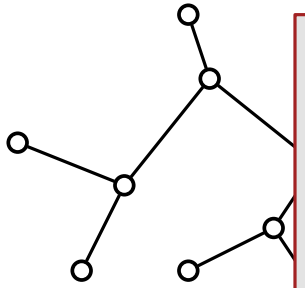
bipartiter Graph

$G = (A \cup B, E)$



In-tree / zur Wurzel ger. Baum

Besondere Graphen



Bäume

Achtung! Zusätzliche Forderung notwendig, damit Vorgänger von v keiner der bereits betrachteten Knoten sein kann:

- entweder kreisfreiheit
- oder: jeder Knoten hat Pfad zur Wurzel

Charakterisierung über $\deg_{\text{out}}$

Ein Knoten r mit $\deg_{\text{out}}(r) = 0$

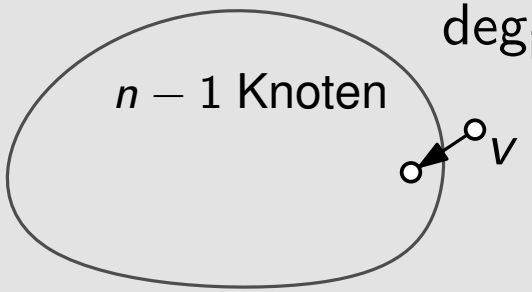
Jeder andere Knoten v:

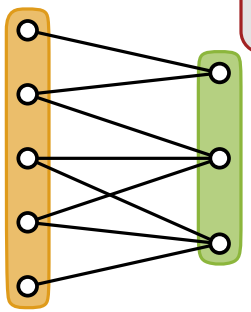
$$\deg_{\text{out}}(v) = 1$$

Beweis: $\Leftarrow$ per Induktion

Schritt: $\deg_{\text{out}}(v) = 1$

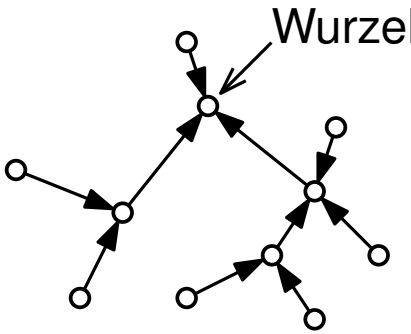
$\deg_{\text{in}}(v) = 0$





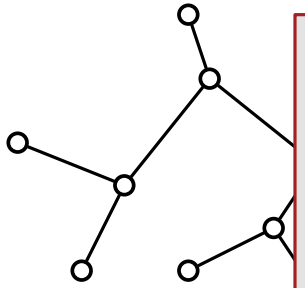
bipartiter Graph

$G = (A \cup B, E)$



In-tree / zur Wurzel ger. Baum

Besondere Graphen



Bäume

Achtung! Zusätzliche Forderung notwendig, damit Vorgänger von v keiner der bereits betrachteten Knoten sein kann:

- entweder kreisfreiheit
- oder: jeder Knoten hat Pfad zur Wurzel

Charakterisierung über $\deg_{\text{out}}$

Ein Knoten r mit $\deg_{\text{out}}(r) = 0$

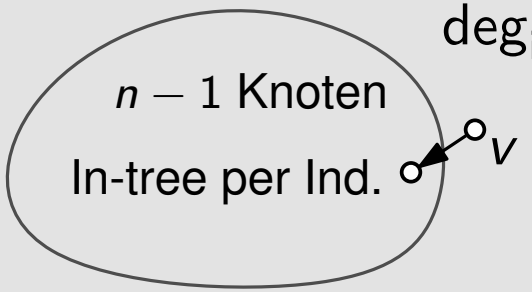
Jeder andere Knoten v:

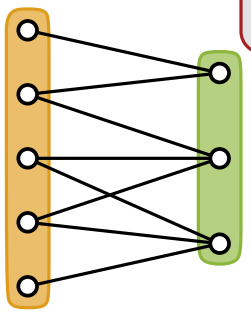
$$\deg_{\text{out}}(v) = 1$$

Beweis: $\Leftarrow$ per Induktion

Schritt: $\deg_{\text{out}}(v) = 1$

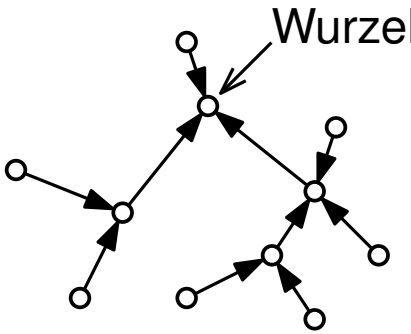
$\deg_{\text{in}}(v) = 0$





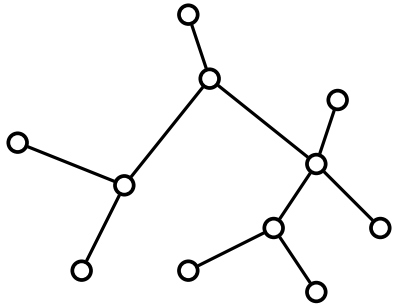
bipartiter Graph

$G = (A \cup B, E)$

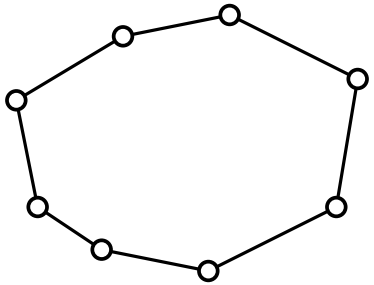


In-tree / zur Wurzel ger. Baum

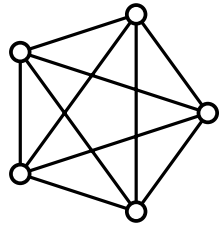
Besondere Graphen



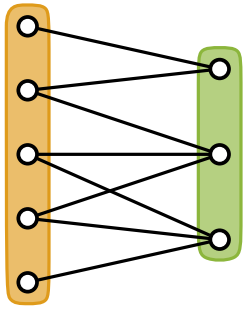
Bäume



Kreise

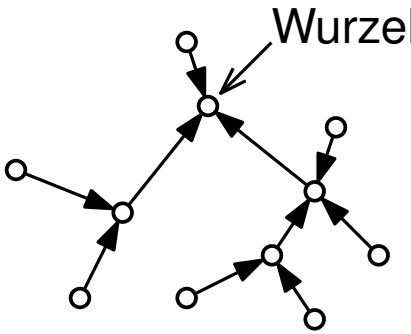


vollst. Graph K_n



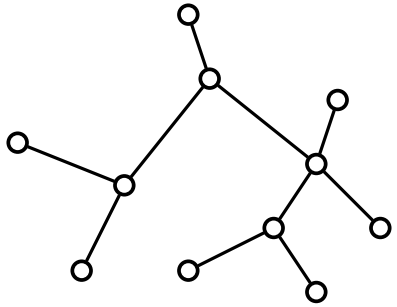
bipartiter Graph

$G = (A \cup B, E)$

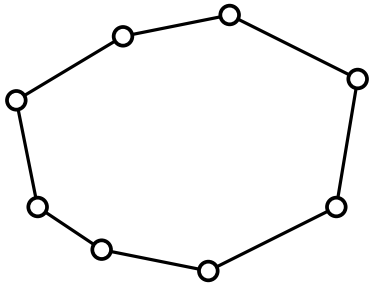


In-tree / zur Wurzel ger. Baum

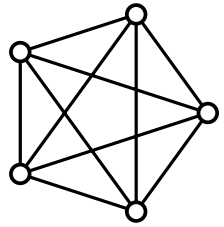
Besondere Graphen



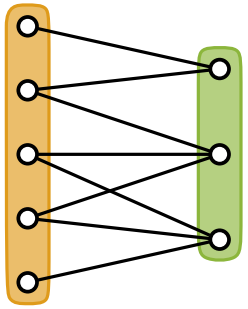
Bäume



Kreise

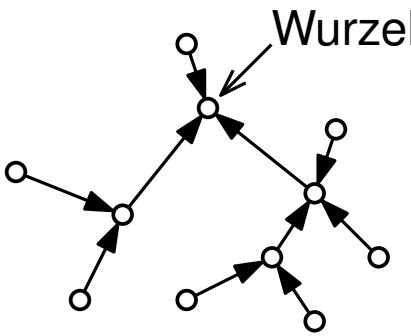


vollst. Graph K_n

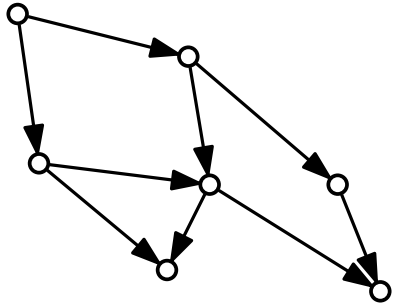


bipartiter Graph

$G = (A \cup B, E)$

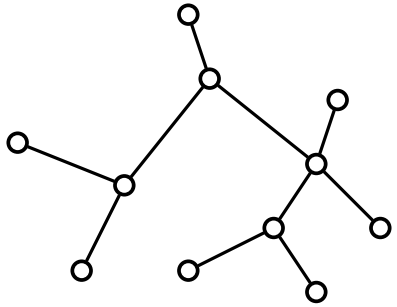


In-tree / zur Wurzel ger. Baum

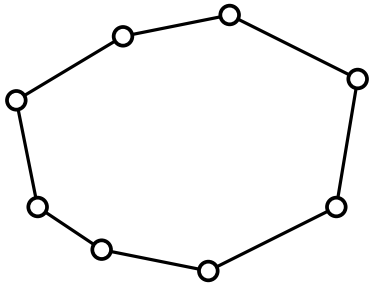


directed acyclic graph (DAG)

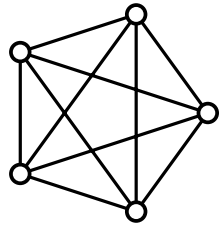
Besondere Graphen



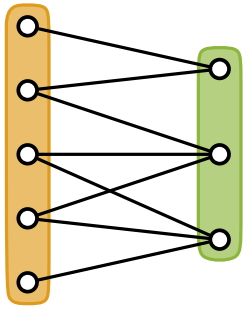
Bäume



Kreise

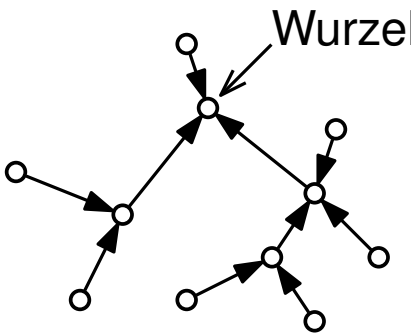


vollst. Graph K_n

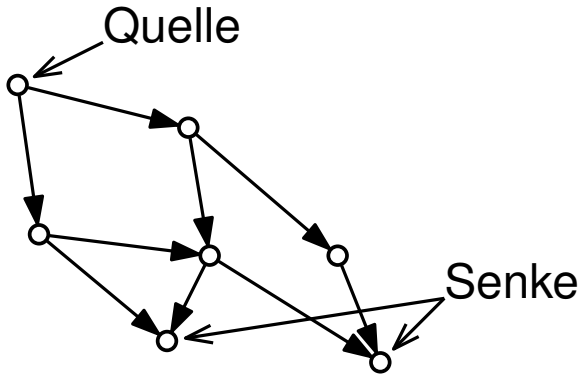


bipartiter Graph

$G = (A \cup B, E)$



In-tree / zur Wurzel ger. Baum



directed acyclic graph (DAG)

Problem: Dependencies

Problem: Dependencies

Package Details: ipe 7.2.24-3

Git Clone URL:	https://aur.archlinux.org/ipe.git (read-only, click to copy)
Package Base:	ipe
Description:	The extensible drawing editor
Upstream URL:	http://ipe.otfried.org/
Licenses:	GPL
Conflicts:	ipe
Submitter:	foxcub
Maintainer:	foxcub
Last Packager:	foxcub
Votes:	77
Popularity:	0.033711
First Submitted:	2007-01-02 01:58 (UTC)
Last Updated:	2021-06-16 18:14 (UTC)

Dependencies (9)

freetype2 (*freetype2-minimal-git*, *freetype2-ttmetrics*, *freetype2-v35*, *freetype2-git*, *freetype2-ultimate5*, *freetype2-infinality-remix*)
gsl (*gsl-git*)
hicolor-icon-theme (*hicolor-icon-theme-git*)
libspiro
lua53
poppler (*poppler-minimal*, *poppler-lcdfilter*, *poppler-lcd*, *poppler-git*)
qt5-base (*qt5-base-git*, *qt5-base-headless*)
qt5-svg (*qt5-svg-git*)
zlib (*zlib-static*, *zlib-git*, *zlib-asm*, *minizip-asm*, *zlib-ng*, *zlib-ng-compat-git*)

Required by (4)

cgal-ipelets
cgal-ipelets (*make*)
ipe-tools-git
ipe2tikz-git

Problem: Dependencies

Package Details: ipe 7.2.24-3

Git Clone URL: https://aur.archlinux.org/ipe.git (read-only, click to copy)

Packag
Descrip
Upstrea
License
Conflict
Submitt
Maintai
Last Pa
Votes:
Popular
First Su
Last Up

Depen

freetyp
freetyp
gsl (*gsl*
hicolor-
libspiro
lua53
poppler
qt5-bas
qt5-svg
zlib (*zli*
git)

poppler 22.06.0-1

Architecture:	x86_64
Repository:	Extra
Split Packages:	poppler-glib, poppler-qt5, poppler-qt6
Description:	PDF rendering library based on xpdf 3.0
Upstream URL:	https://poppler.freedesktop.org/
License(s):	GPL
Provides:	libpoppler-cpp.so=0-64, libpoppler.so=122-64
Conflicts:	poppler-qt3<22.06.0, poppler-qt4<22.06.0
Maintainers:	Andreas Radke
Package Size:	1.5 MB
Installed Size:	6.1 MB
Last Packager:	Andreas Radke
Build Date:	2022-06-02 17:59 UTC
Signed By:	Andreas Radke
Signature Date:	2022-06-02 18:03 UTC
Last Updated:	2022-06-04 07:20 UTC

Dependencies (29)

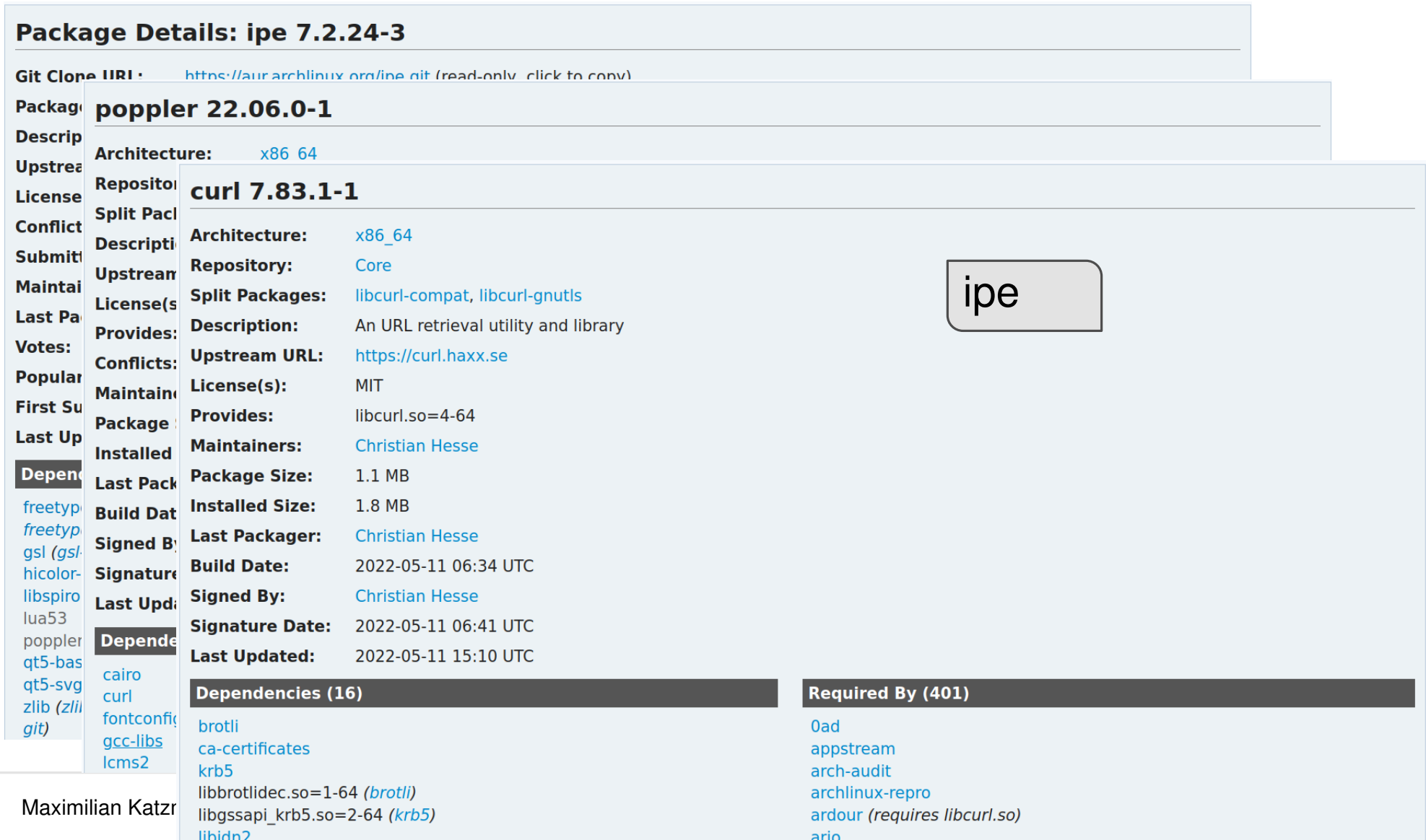
cairo
curl
fontconfig
gcc-libs
lcms2

Required By (45)

auto-multiple-choice
cups-filters
deepin-file-manager
docparser
efl

Problem: Dependencies

Problem: Dependencies



Problem: Dependencies

Package Details: ipe 7.2.24-3

Git Clone URL: https://aur.archlinux.org/ipe.git (read-only, click to copy)

Package

Descrip

Upstrea

License

Conflict

Submitt

Maintai

Last Pa

Votes:

Popular

First Su

Last Up

Depen

freetyp
freetyp
gsl *(gsl*
hicolor-
libspiro
lua53
poppler
qt5-bas
qt5-svg
zlib *(zli*
git)

poppler 22.06.0-1

Architecture: x86_64

Repositor

Split Pacl

Descripti

Upstream

License(s

Provides:

Conflicts:

Maintain

Package

Installed

Last Pack

Build Dat

Signed By

Signature

Last Upd

Depende

cairo
curl
fontconfig
gcc-libs
lcms2

curl 7.83.1-1

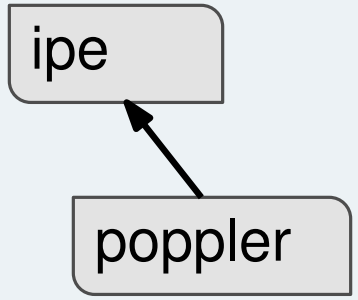
Architecture:	x86_64
Repository:	Core
Split Packages:	libcurl-compat, libcurl-gnutls
Description:	An URL retrieval utility and library
Upstream URL:	https://curl.haxx.se
License(s):	MIT
Provides:	libcurl.so=4-64
Maintainers:	Christian Hesse
Package Size:	1.1 MB
Installed Size:	1.8 MB
Last Packager:	Christian Hesse
Build Date:	2022-05-11 06:34 UTC
Signed By:	Christian Hesse
Signature Date:	2022-05-11 06:41 UTC
Last Updated:	2022-05-11 15:10 UTC

Dependencies (16)

brotli
ca-certificates
krb5
libbrotlidec.so=1-64 *(brotli)*
libgssapi_krb5.so=2-64 *(krb5)*
libidn2

Required By (401)

0ad
appstream
arch-audit
archlinux-repro
ardour *(requires libcurl.so)*
ario



Problem: Dependencies

Package Details: ipe 7.2.24-3

Git Clone URL: https://aur.archlinux.org/ipe.git (read-only, click to copy)

Package

Descrip

Upstrea

License

Conflict

Submitt

Maintai

Last Pa

Votes:

Popular

First Su

Last Up

Depen

freetyp
freetyp
gsl *(gsl*
hicolor-
libspiro
lua53
poppler
qt5-bas
qt5-svg
zlib *(zli*
git)

poppler 22.06.0-1

Architecture: x86_64

Repositor

Split Pacl

Descripti

Upstream

License(s

Provides:

Conflicts:

Maintain

Package

Installed

Last Pack

Build Dat

Signed By

Signature

Last Upd

Depende

cairo
curl
fontconfig
gcc-libs
lcms2

curl 7.83.1-1

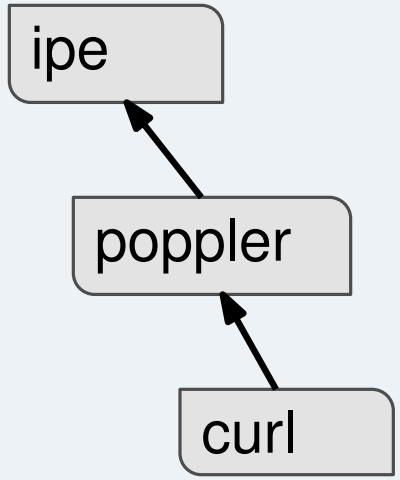
Architecture:	x86_64
Repository:	Core
Split Packages:	libcurl-compat, libcurl-gnutls
Description:	An URL retrieval utility and library
Upstream URL:	https://curl.haxx.se
License(s):	MIT
Provides:	libcurl.so=4-64
Maintainers:	Christian Hesse
Package Size:	1.1 MB
Installed Size:	1.8 MB
Last Packager:	Christian Hesse
Build Date:	2022-05-11 06:34 UTC
Signed By:	Christian Hesse
Signature Date:	2022-05-11 06:41 UTC
Last Updated:	2022-05-11 15:10 UTC

Dependencies (16)

brotli
ca-certificates
krb5
libbrotlidec.so=1-64 *(brotli)*
libgssapi_krb5.so=2-64 *(krb5)*
libidn2

Required By (401)

0ad
appstream
arch-audit
archlinux-repro
ardour *(requires libcurl.so)*
ario



Problem: Dependencies

Package Details: ipe 7.2.24-3

Git Clone URL: https://aur.archlinux.org/ipe.git (read-only, click to copy)

poppler 22.06.0-1

Architecture: x86_64

curl 7.83.1-1

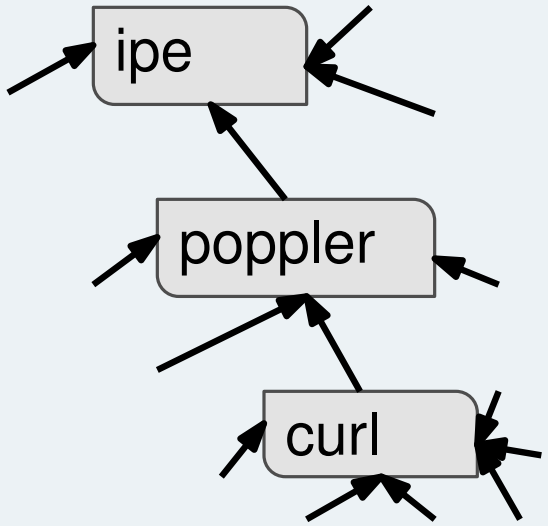
Architecture:	x86_64
Repository:	Core
Split Packages:	libcurl-compat, libcurl-gnutls
Description:	An URL retrieval utility and library
Upstream URL:	https://curl.haxx.se
License(s):	MIT
Provides:	libcurl.so=4-64
Maintainers:	Christian Hesse
Package Size:	1.1 MB
Installed Size:	1.8 MB
Last Packager:	Christian Hesse
Build Date:	2022-05-11 06:34 UTC
Signed By:	Christian Hesse
Signature Date:	2022-05-11 06:41 UTC
Last Updated:	2022-05-11 15:10 UTC

Dependencies (16)

brotli
ca-certificates
krb5
libbrotlidec.so=1-64 *(brotli)*
libgssapi_krb5.so=2-64 *(krb5)*

Required By (401)

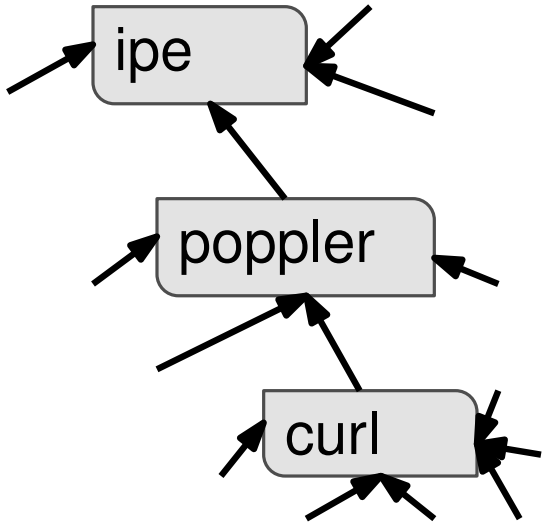
0ad
appstream
arch-audit
archlinux-repro
ardour *(requires libcurl.so)*



Problem: Dependencies

Modellierung als Graph

- Knoten V: Menge von Paketen
- Knoten E: $(v, w) \in E \Leftrightarrow v$ von w benötigt



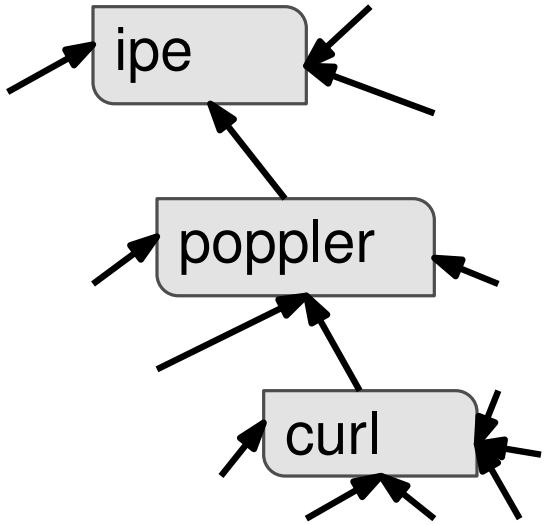
Problem: Dependencies

Modellierung als Graph

- Knoten V: Menge von Paketen
- Knoten E: $(v, w) \in E \Leftrightarrow v$ von w benötigt

Frage

Gibt es zyklische Abhängigkeiten?



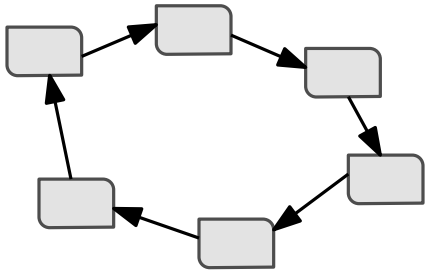
Problem: Dependencies

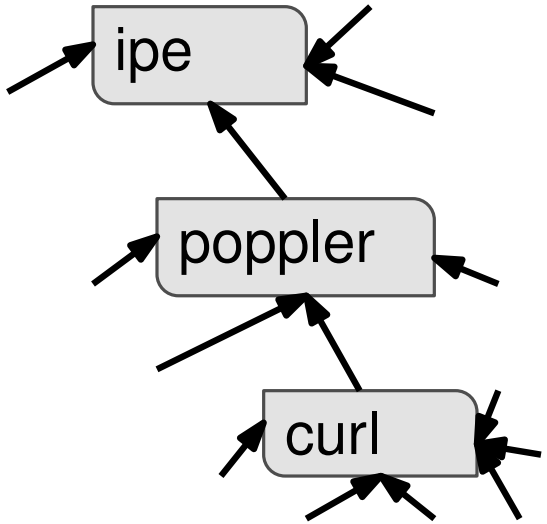
Modellierung als Graph

- Knoten V: Menge von Paketen
- Knoten E: $(v, w) \in E \Leftrightarrow v$ von w benötigt

Frage

Gibt es zyklische Abhängigkeiten?

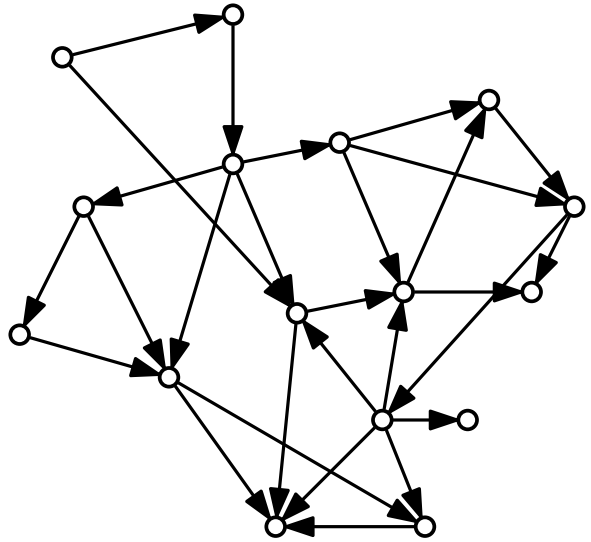




Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

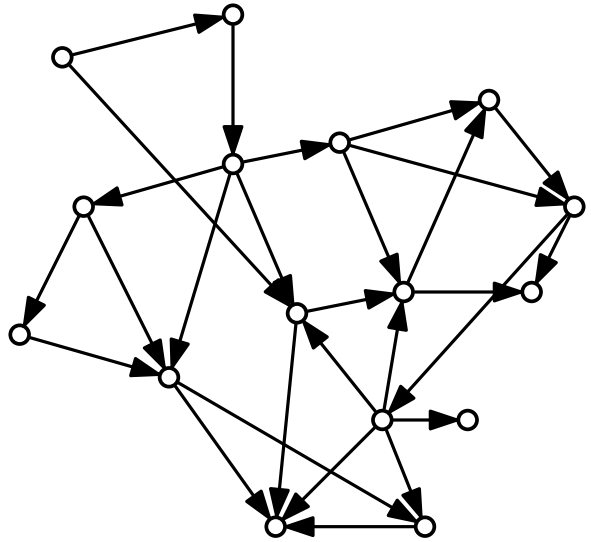


Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze



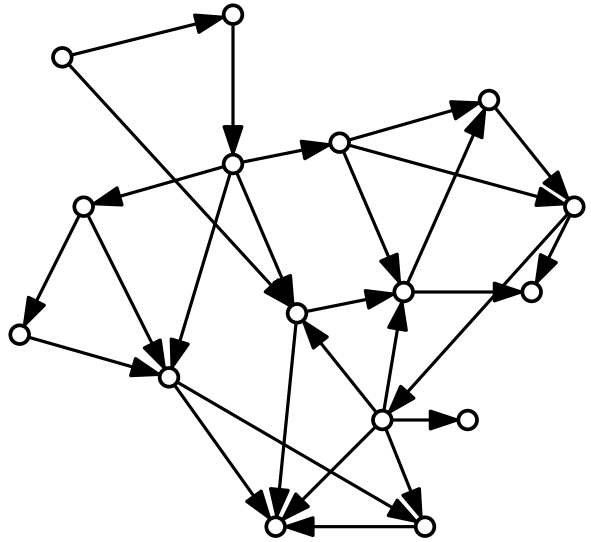
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze

- brute-force



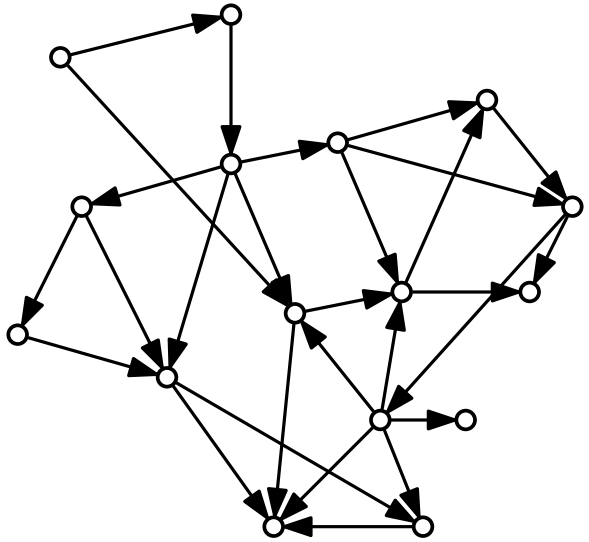
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze

- brute-force
  - betrachte jede Teilmenge $E' \subset E$
    - prüfe ob E' Kreis ergibt



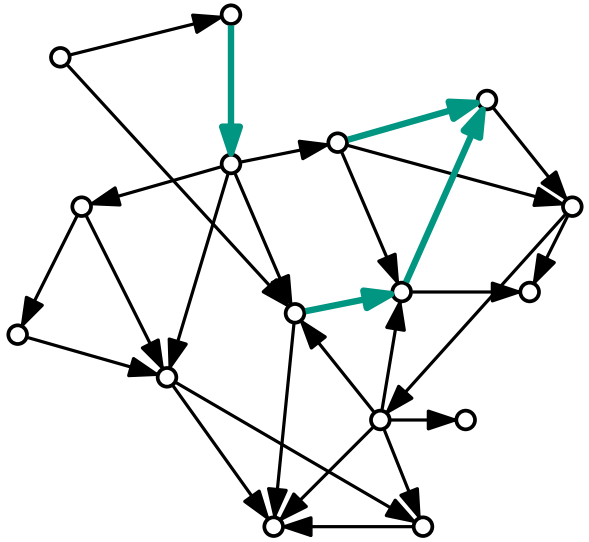
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze

- brute-force
  - betrachte jede Teilmenge $E' \subset E$
    - prüfe ob E' Kreis ergibt



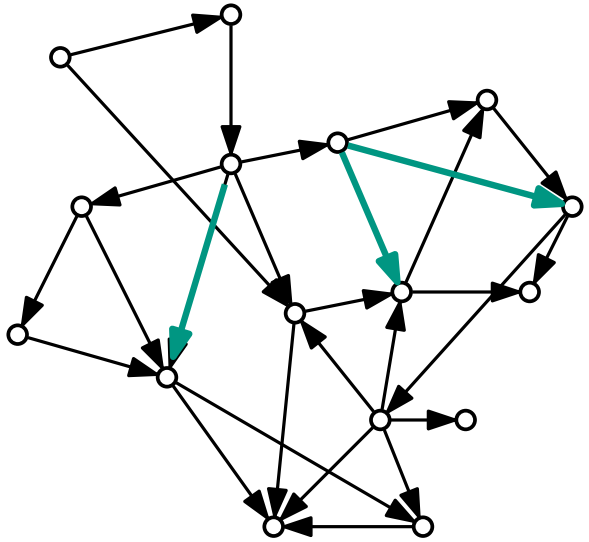
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze

- brute-force
  - betrachte jede Teilmenge $E' \subset E$
    - prüfe ob E' Kreis ergibt



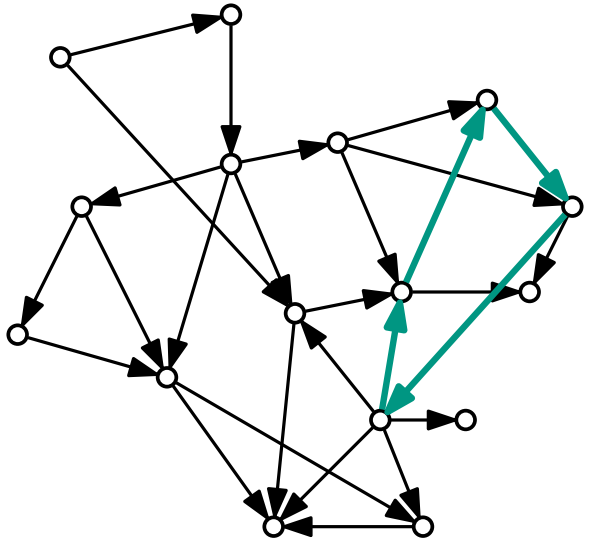
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze

- brute-force
  - betrachte jede Teilmenge $E' \subset E$
    - prüfe ob E' Kreis ergibt



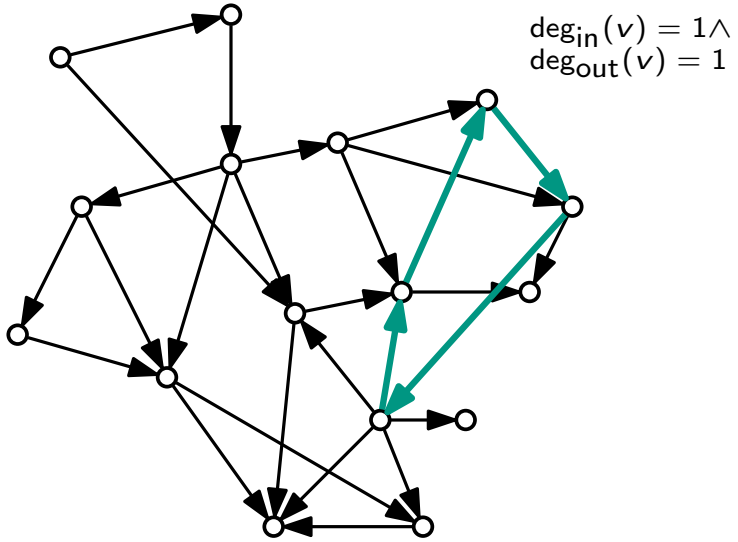
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze

- brute-force
  - betrachte jede Teilmenge $E' \subset E$
    - prüfe ob E' Kreis ergibt



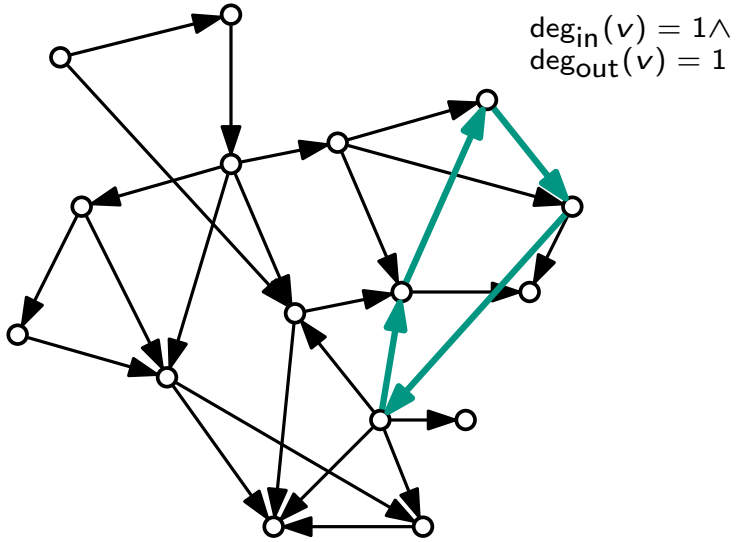
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze

- brute-force
  - betrachte jede Teilmenge $E' \subset E$ $\quad O(2^m)$
    - prüfe ob E' Kreis ergibt



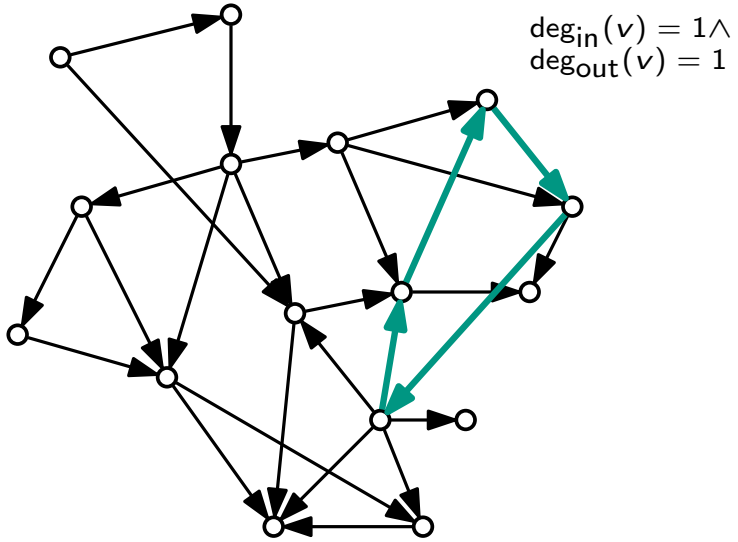
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze

- brute-force
  - betrachte jede Teilmenge $E' \subset E$ $O(2^m)$
    - prüfe ob E' Kreis ergibt $O(n + m)$



Cyclic Dependency

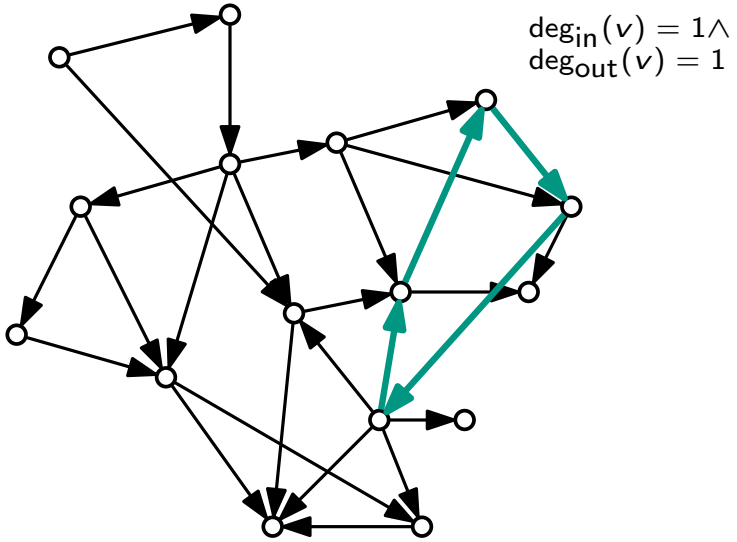
Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze

- brute-force
  - betrachte jede Teilmenge $E' \subset E$ $O(2^m)$
    - prüfe ob E' Kreis ergibt $O(n + m)$

Gesamt: $O(2^m \cdot (n+m))$



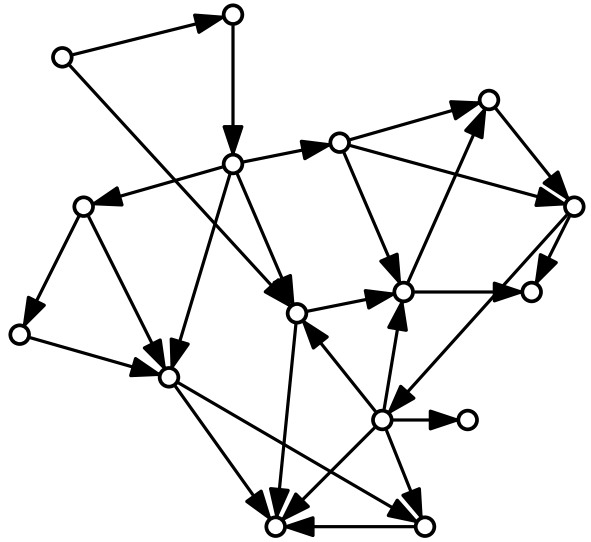
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze

- brute-force



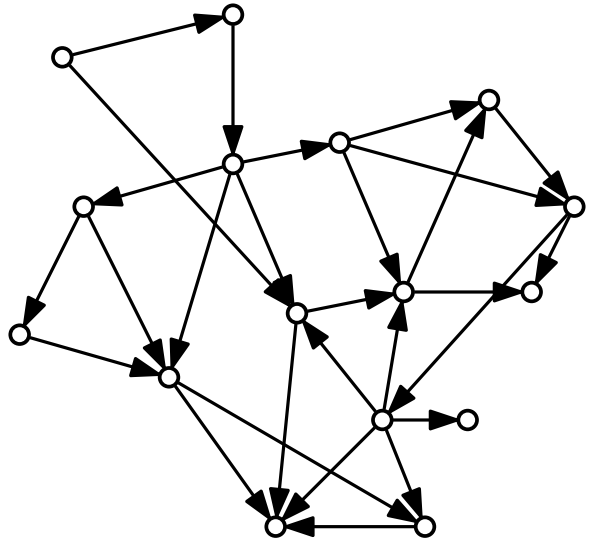
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze

- brute-force
- etwas geschickter



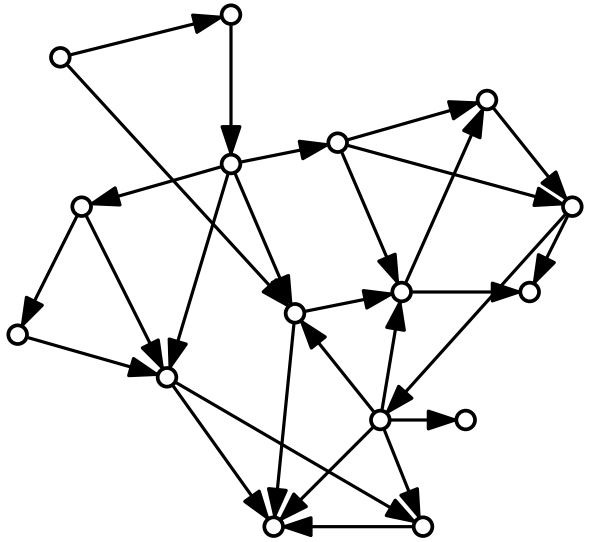
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze

- brute-force
- etwas geschickter
  - betrachte jede Kante $(a, b) \in E$



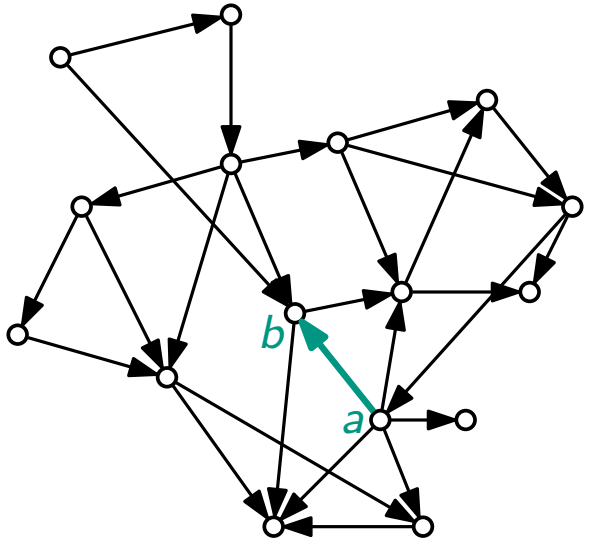
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze

- brute-force
- etwas geschickter
  - betrachte jede Kante $(a, b) \in E$



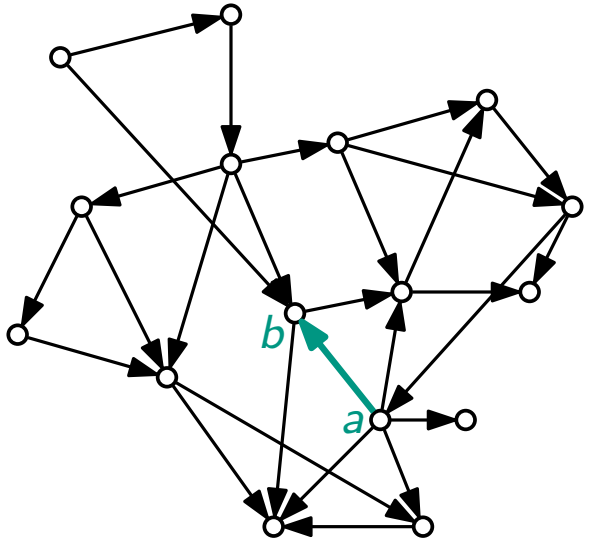
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze

- brute-force
- etwas geschickter
  - betrachte jede Kante $(a, b) \in E$
    - suche b-a-Pfad



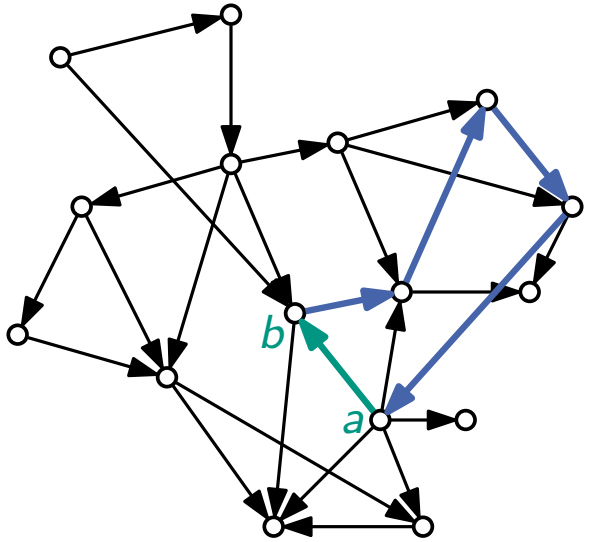
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze

- brute-force
- etwas geschickter
  - betrachte jede Kante $(a, b) \in E$
    - suche b-a-Pfad



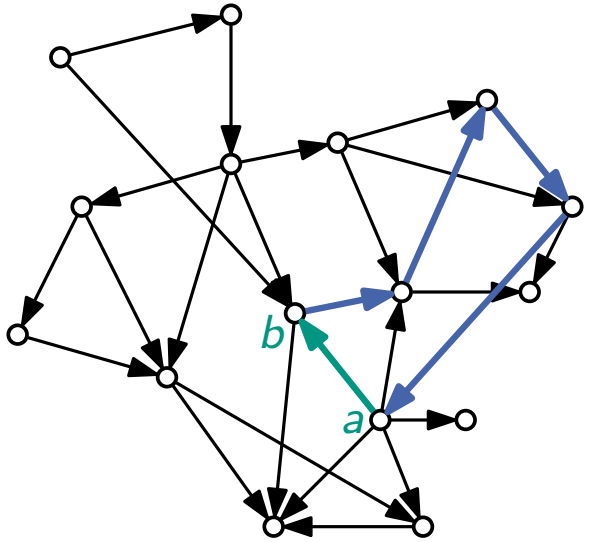
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze

- brute-force
- etwas geschickter
  - betrachte jede Kante $(a, b) \in E$ $\quad O(m)$
    - suche b-a-Pfad



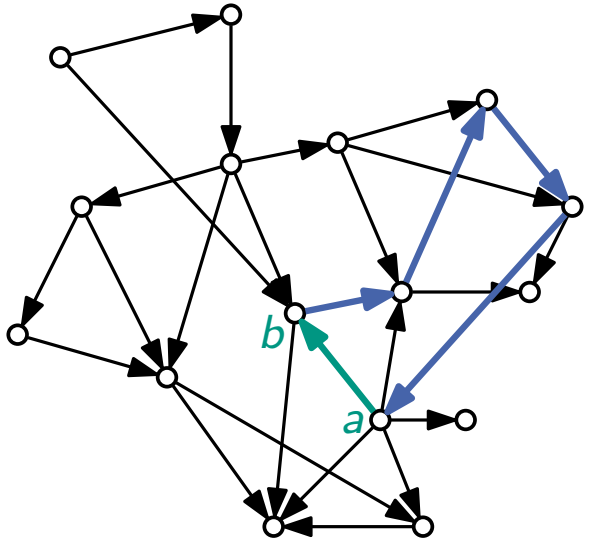
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze

- brute-force
- etwas geschickter
  - betrachte jede Kante $(a, b) \in E$ $O(m)$
    - suche b-a-Pfad $O(n + m)$



Cyclic Dependency

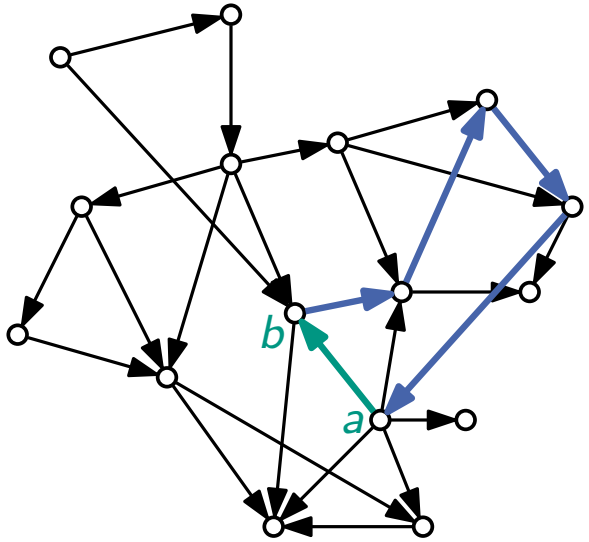
Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze

- brute-force
- etwas geschickter
  - betrachte jede Kante $(a, b) \in E$ $O(m)$
    - suche b-a-Pfad $O(n + m)$

Gesamt: $O(m(n+m))$



Cyclic Dependency

Problemstellung

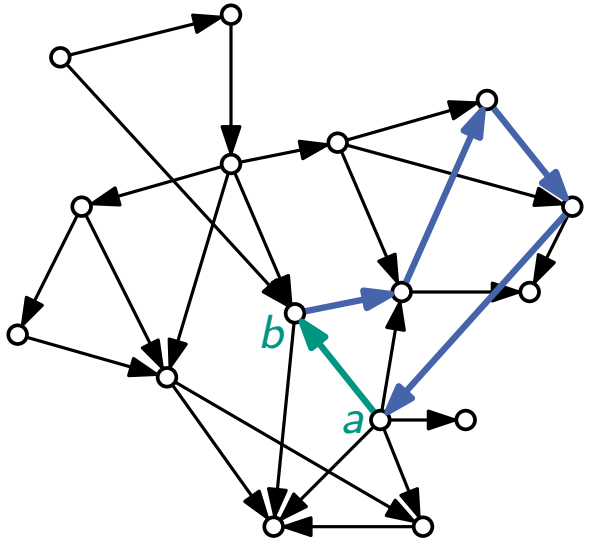
- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze

- brute-force
- etwas geschickter
  - betrachte jede Kante $(a, b) \in E$ $O(m)$
    - suche b-a-Pfad $O(n + m)$

Gesamt: $O(m(n+m))$

Frage: Ist Linearzeit möglich?

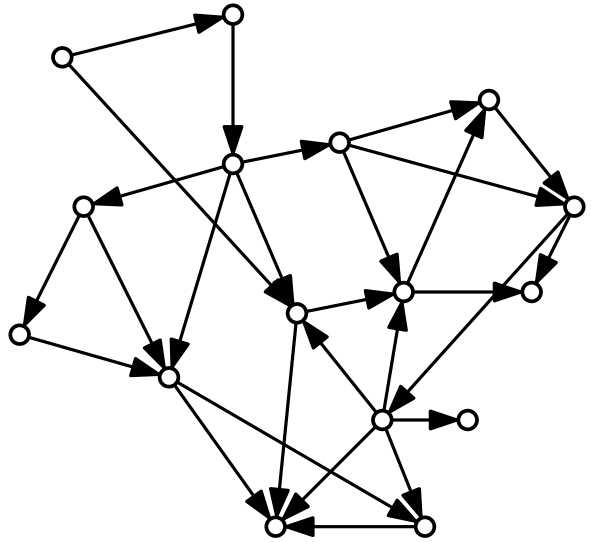


Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?



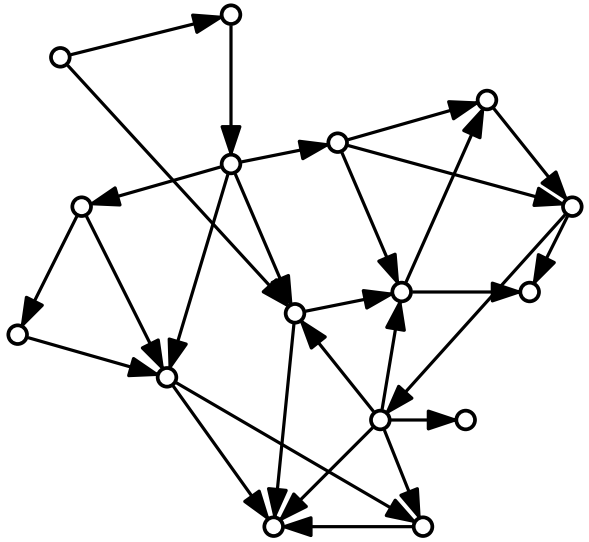
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke



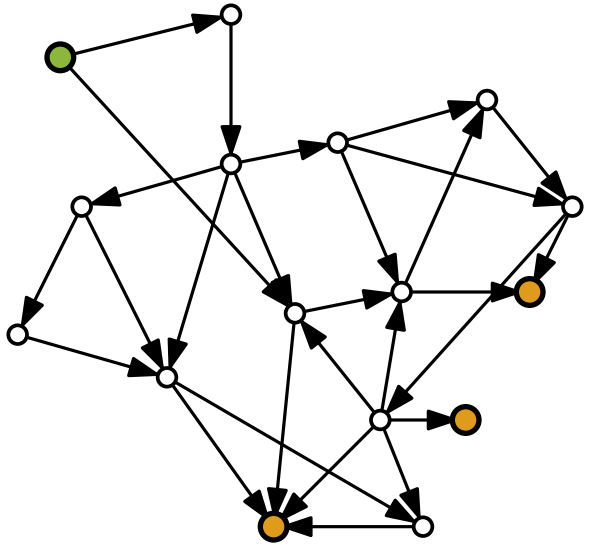
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke



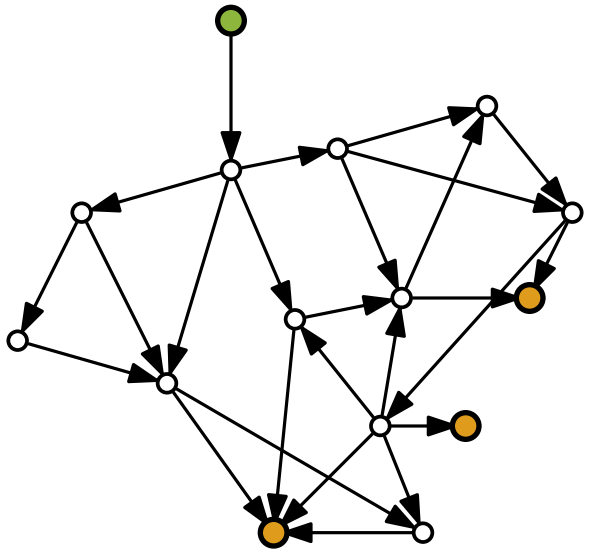
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke



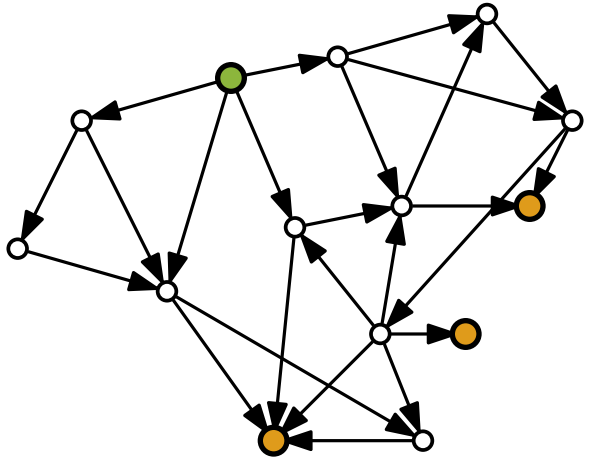
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke



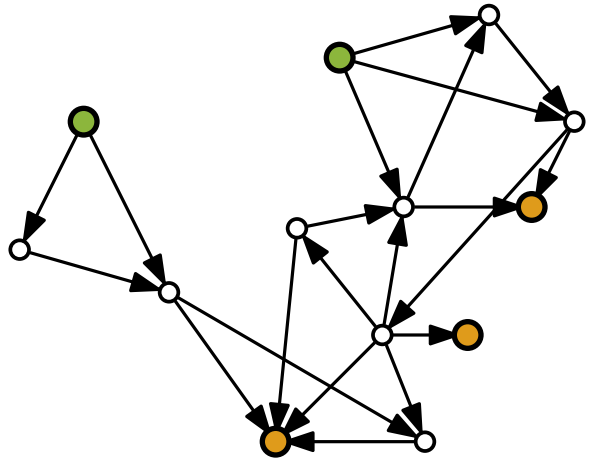
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke



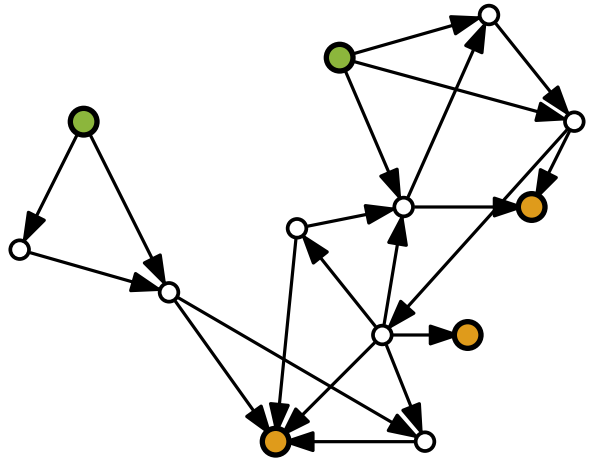
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke
  - Löschen von Quellen/Senken erhält Kreis(freiheit)



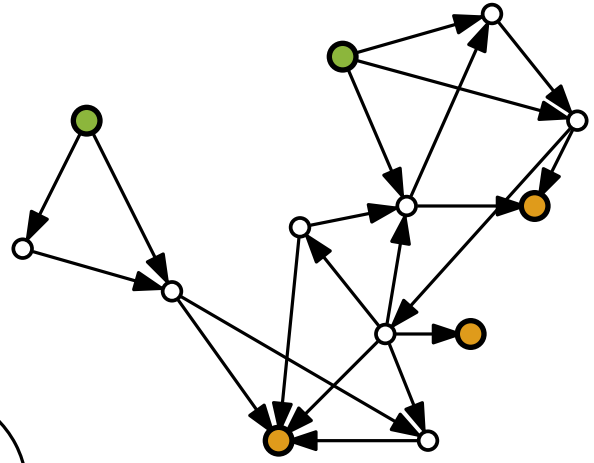
Cyclic Dependency

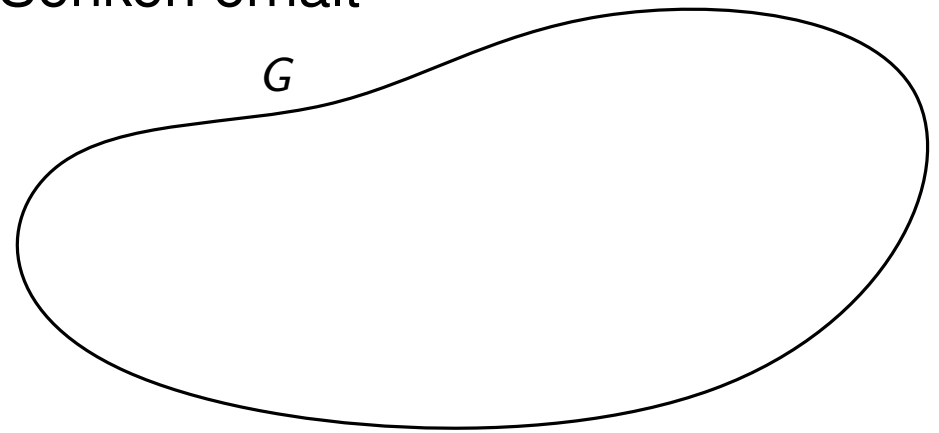
Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke
  - Löschen von Quellen/Senken erhält Kreis(freiheit)





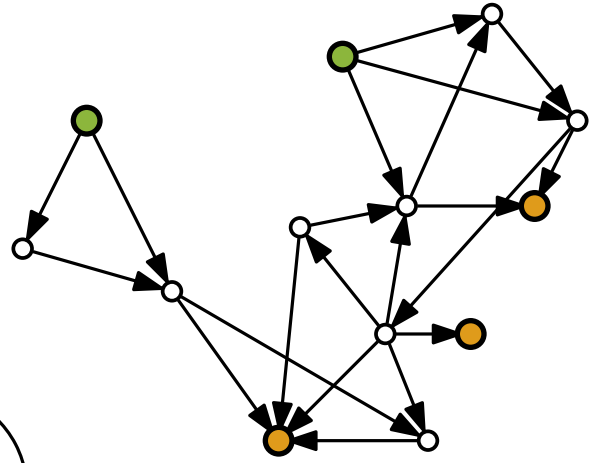
Cyclic Dependency

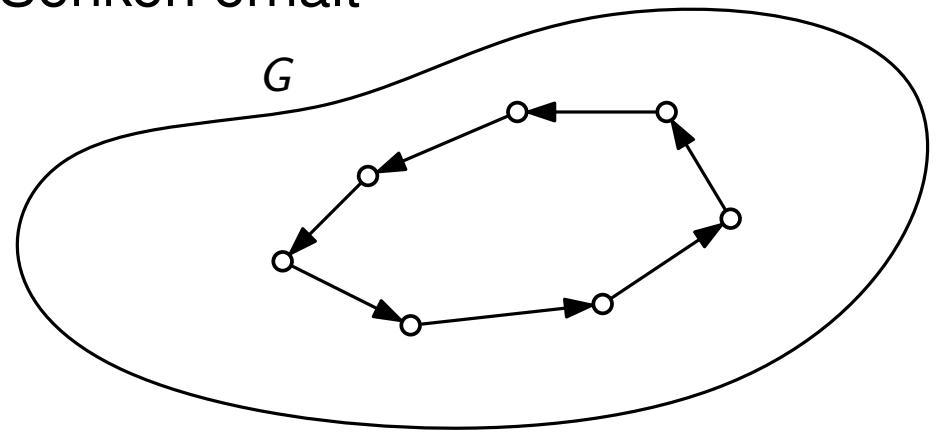
Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke
  - Löschen von Quellen/Senken erhält Kreis(freiheit)





Cyclic Dependency

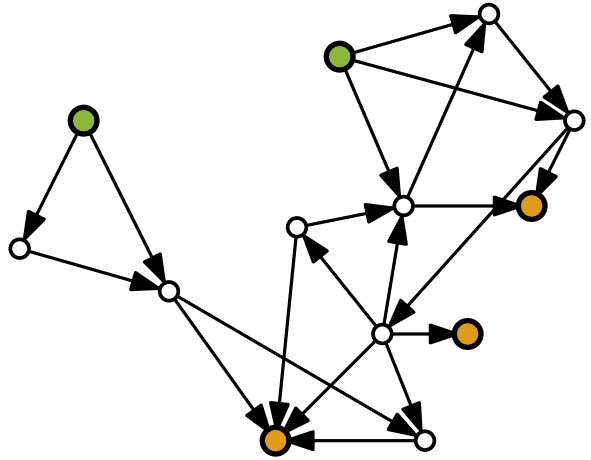
Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke
  - Löschen von Quellen/Senken erhält Kreis(freiheit)

Effiziente Implementierung



Cyclic Dependency

Problemstellung

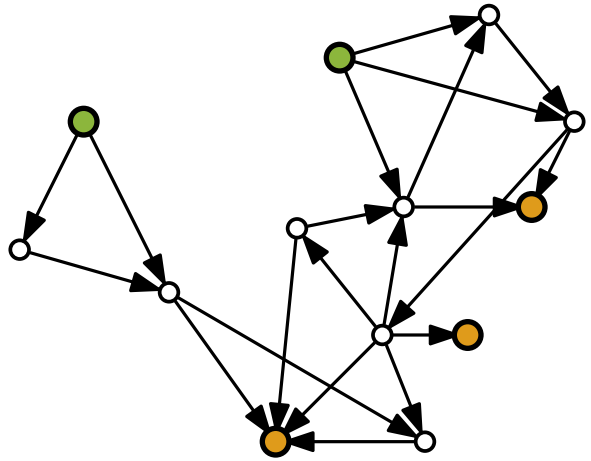
- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke
  - Löschen von Quellen/Senken erhält Kreis(freiheit)

Effiziente Implementierung

- finde zu Beginn alle Quellen



Cyclic Dependency

Problemstellung

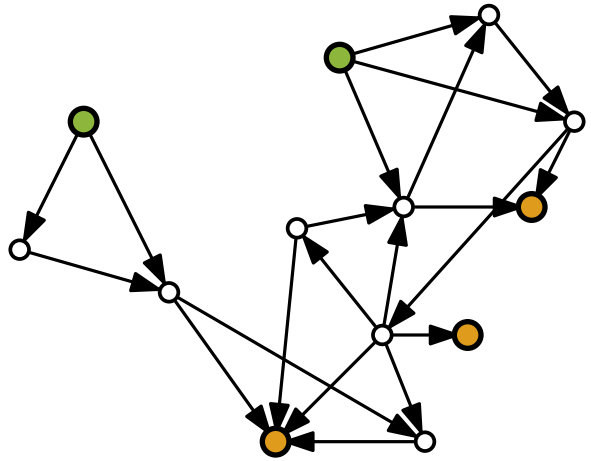
- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke
  - Löschen von Quellen/Senken erhält Kreis(freiheit)

Effiziente Implementierung

- finde zu Beginn alle Quellen $\Theta(n + m)$



Cyclic Dependency

Problemstellung

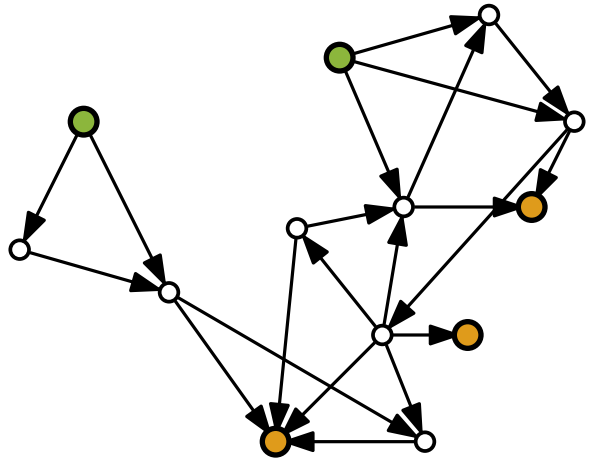
- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke
  - Löschen von Quellen/Senken erhält Kreis(freiheit)

Effiziente Implementierung

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q



Cyclic Dependency

Problemstellung

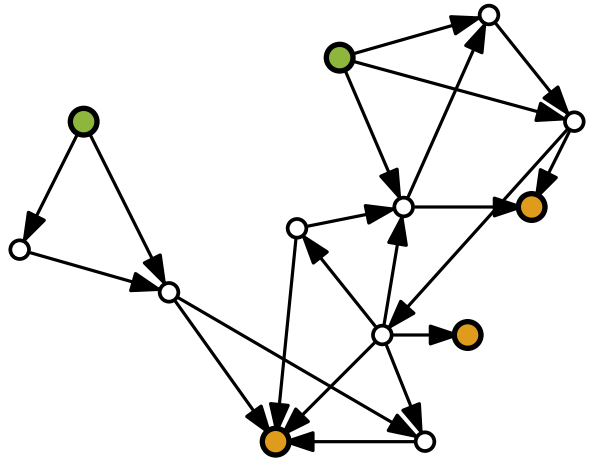
- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke
  - Löschen von Quellen/Senken erhält Kreis(freiheit)

Effiziente Implementierung

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q
  - suche neue Quellen in $N(q)$



Cyclic Dependency

Problemstellung

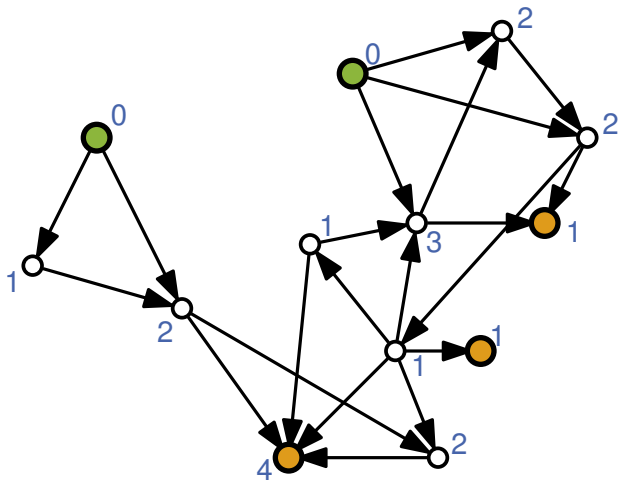
- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke
  - Löschen von Quellen/Senken erhält Kreis(freiheit)

Effiziente Implementierung

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q
  - suche neue Quellen in $N(q)$



Cyclic Dependency

Problemstellung

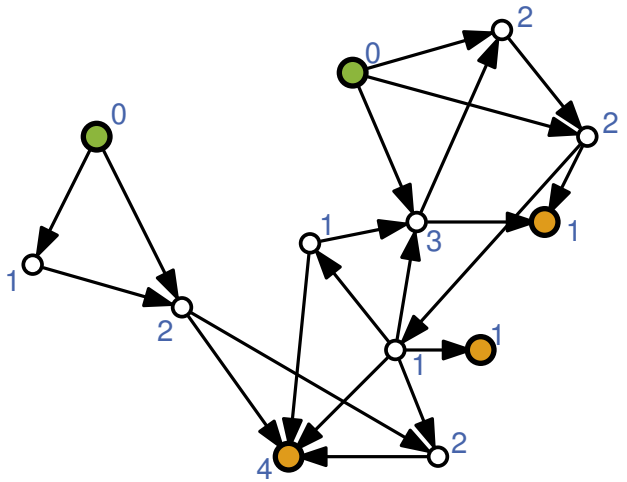
- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke
  - Löschen von Quellen/Senken erhält Kreis(freiheit)

Effiziente Implementierung

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q
  - suche neue Quellen in $N(q)$



$\deg_{\text{in}} : [\mathbb{N}] = [0, 0, 2, 1, \ldots]$

Cyclic Dependency

Problemstellung

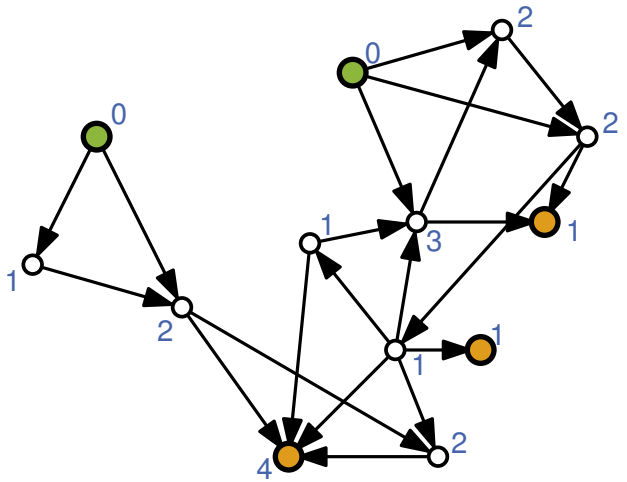
- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke
  - Löschen von Quellen/Senken erhält Kreis(freiheit)

Effiziente Implementierung

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q
  - suche neue Quellen in $N(q)$



$\deg_{\text{in}} : [\mathbb{N}] = [0, 0, 2, 1, \ldots]$

sources : Queue

Cyclic Dependency

Problemstellung

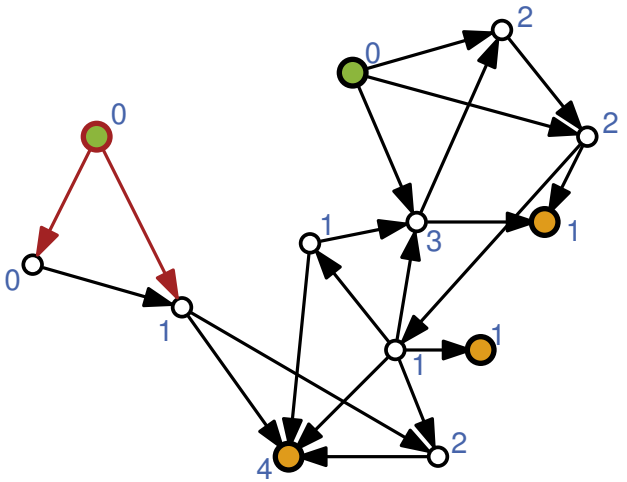
- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke
  - Löschen von Quellen/Senken erhält Kreis(freiheit)

Effiziente Implementierung

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q
  - suche neue Quellen in $N(q)$



$\deg_{\text{in}} : [\mathbb{N}] = [0, 0, 2, 1, \ldots]$

sources : Queue

Cyclic Dependency

Problemstellung

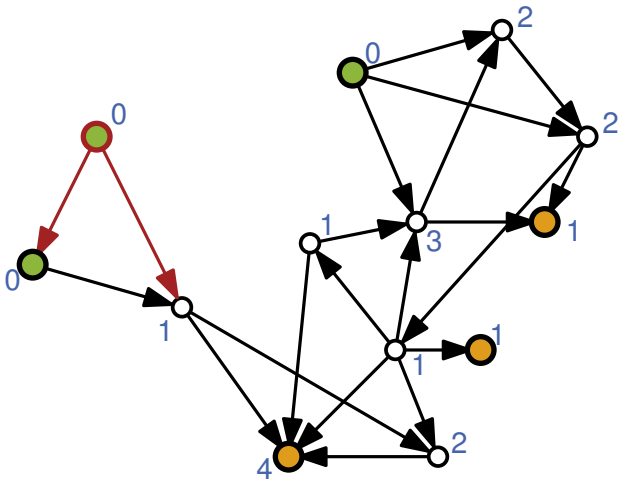
- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke
  - Löschen von Quellen/Senken erhält Kreis(freiheit)

Effiziente Implementierung

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q
  - suche neue Quellen in $N(q)$



$\deg_{\text{in}} : [\mathbb{N}] = [0, 0, 2, 1, \ldots]$

sources : Queue

Cyclic Dependency

Problemstellung

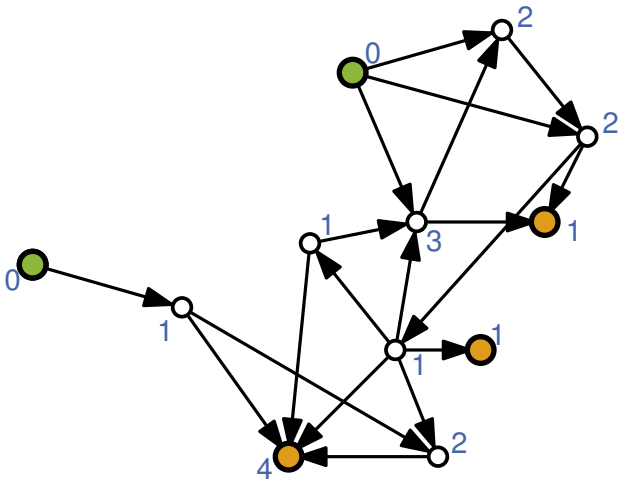
- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke
  - Löschen von Quellen/Senken erhält Kreis(freiheit)

Effiziente Implementierung

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q
  - suche neue Quellen in $N(q)$



$\deg_{\text{in}} : [\mathbb{N}] = [0, 0, 2, 1, \ldots]$

sources : Queue

Cyclic Dependency

Problemstellung

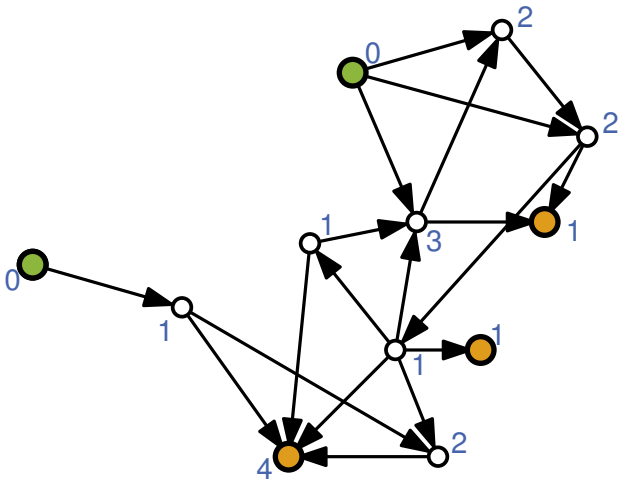
- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke
  - Löschen von Quellen/Senken erhält Kreis(freiheit)

Effiziente Implementierung

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q
  - suche neue Quellen in $N(q)$ $\Theta(\deg_{\text{out}}(v))$



$\deg_{\text{in}} : [\mathbb{N}] = [0, 0, 2, 1, \ldots]$

sources : Queue

Cyclic Dependency

Problemstellung

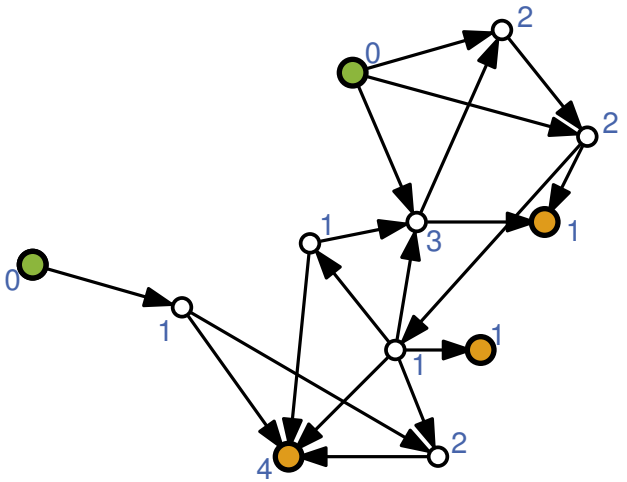
- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke
  - Löschen von Quellen/Senken erhält Kreis(freiheit)

Effiziente Implementierung

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q $\Theta(\deg_{\text{out}}(v))$?
  - suche neue Quellen in $N(q)$ $\Theta(\deg_{\text{out}}(v))$



$\deg_{\text{in}} : [\mathbb{N}] = [0, 0, 2, 1, \ldots]$

sources : Queue

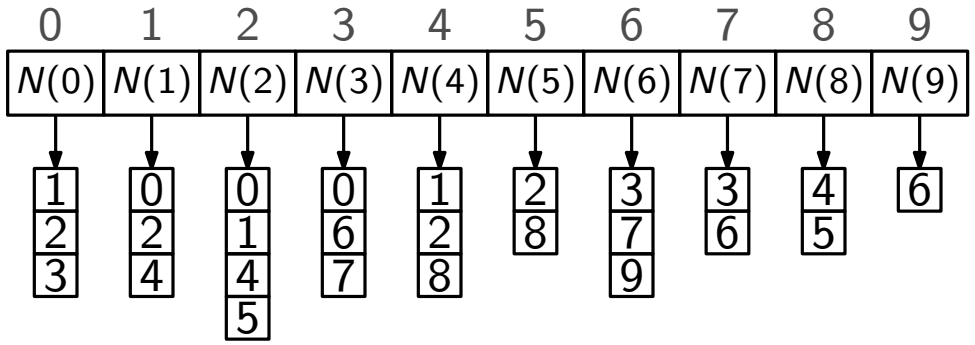
Wiederholung: Repräsentation von Graphen

Adjazenzliste

Adjazenzmatrix

Wiederholung: Repräsentation von Graphen

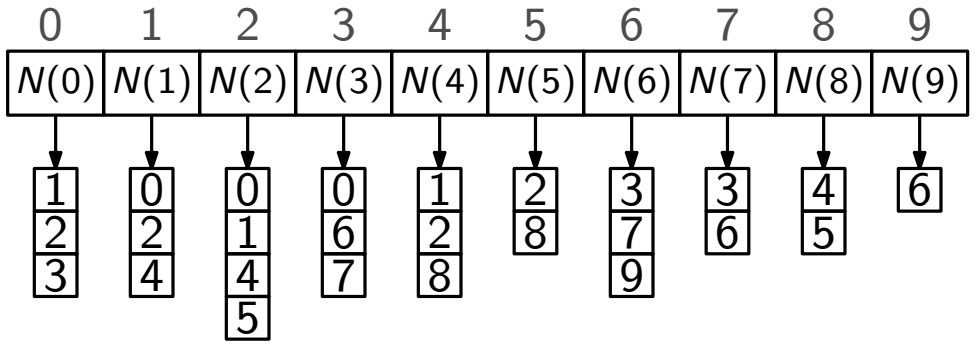
Adjazenzliste



Adjazenzmatrix

Wiederholung: Repräsentation von Graphen

Adjazenzliste

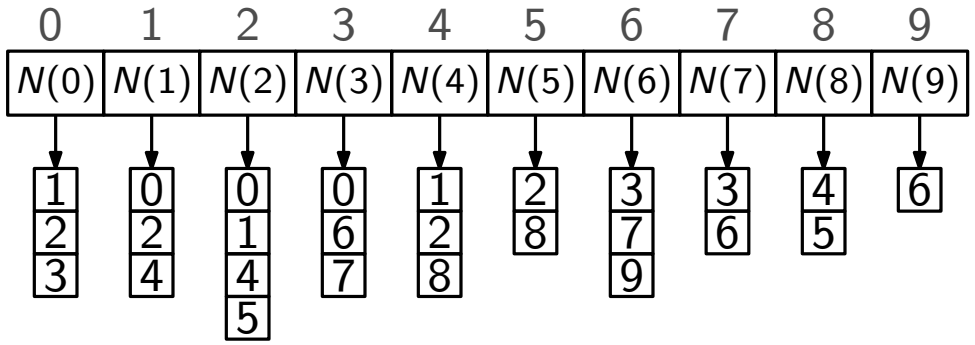


- Array A mit Nachbarschaften als Listen

Adjazenzmatrix

Wiederholung: Repräsentation von Graphen

Adjazenzliste

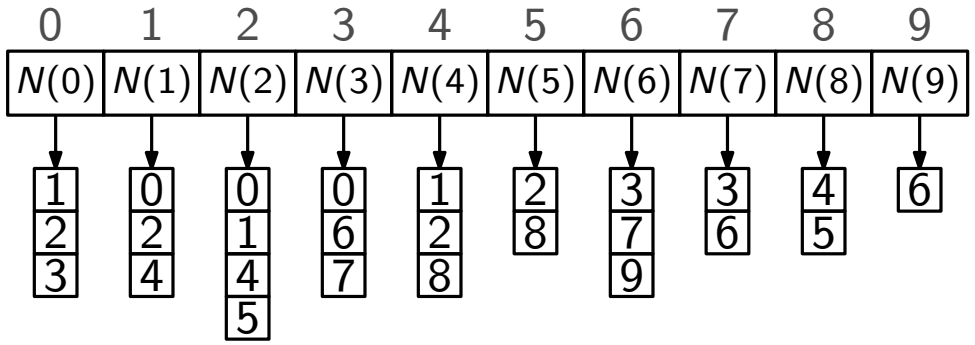


- Array A mit Nachbarschaften als Listen
- über $N(v)$ iterieren: $\Theta(\deg(v))$
- Test ob $\{u, v\} \in E$: $\Theta(\deg(v))$
- $\Theta(n + m)$ Speicher

Adjazenzmatrix

Wiederholung: Repräsentation von Graphen

Adjazenzliste



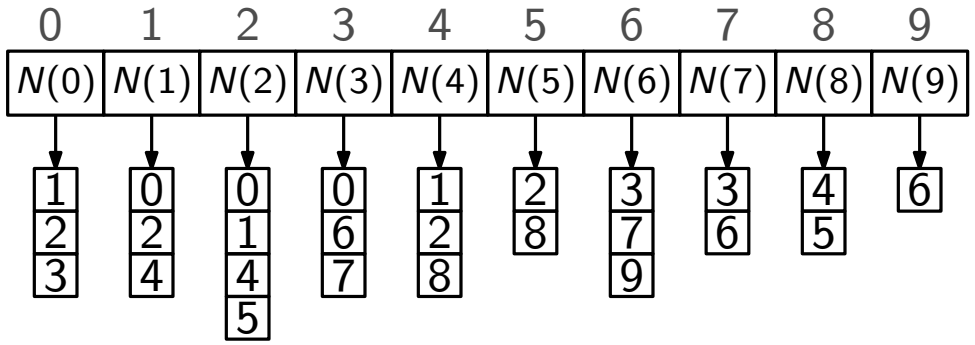
- Array A mit Nachbarschaften als Listen
- über $N(v)$ iterieren: $\Theta(\deg(v))$
- Test ob $\{u, v\} \in E$: $\Theta(\deg(v))$
- $\Theta(n + m)$ Speicher

Adjazenzmatrix

	0	1	2	3	4	5	6	7	8	9
0	0	1	1	1	0	0	0	0	0	0
1	1	0	1	0	1	0	0	0	0	0
2	1	1	0	0	1	1	0	0	0	0
3	1	0	0	0	0	0	1	1	0	0
4	0	1	1	0	0	0	0	0	1	0
5	0	0	1	0	0	0	0	0	1	0
6	0	0	0	1	0	0	0	1	0	1
7	0	0	0	1	0	0	1	0	0	0
8	0	0	0	0	1	1	0	0	0	0
9	0	0	0	0	0	0	1	0	0	0

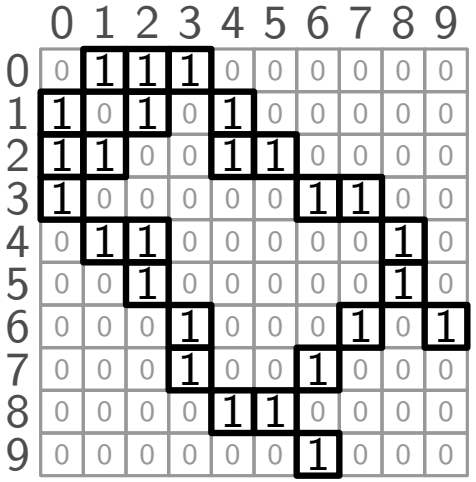
Wiederholung: Repräsentation von Graphen

Adjazenzliste



- Array A mit Nachbarschaften als Listen
- über $N(v)$ iterieren: $\Theta(\deg(v))$
- Test ob $\{u, v\} \in E$: $\Theta(\deg(v))$
- $\Theta(n + m)$ Speicher

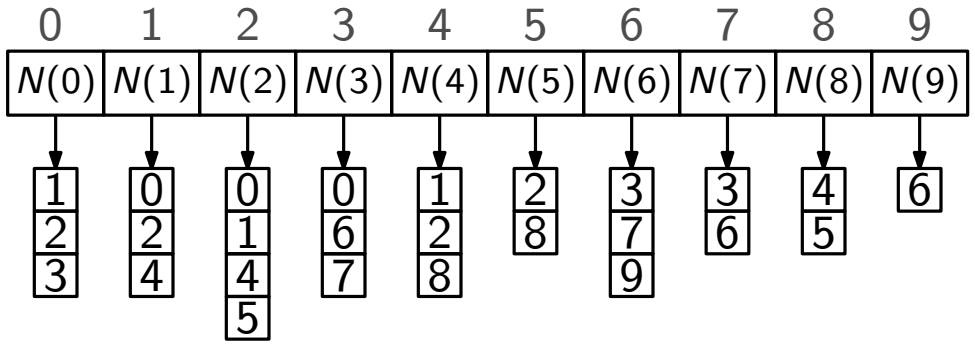
Adjazenzmatrix



- $A[u][v] = 1 \Leftrightarrow \{u, v\} \in E$
- $N(v)$ iterieren: $\Theta(n)$
- $\Theta(n^2)$ Speicher

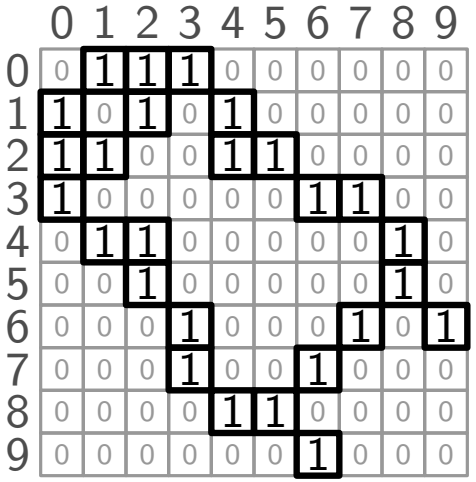
Wiederholung: Repräsentation von Graphen

Adjazenzliste



- Array A mit Nachbarschaften als Listen
- über $N(v)$ iterieren: $\Theta(\deg(v))$
- Test ob $\{u, v\} \in E$: $\Theta(\deg(v))$
- $\Theta(n + m)$ Speicher

Adjazenzmatrix

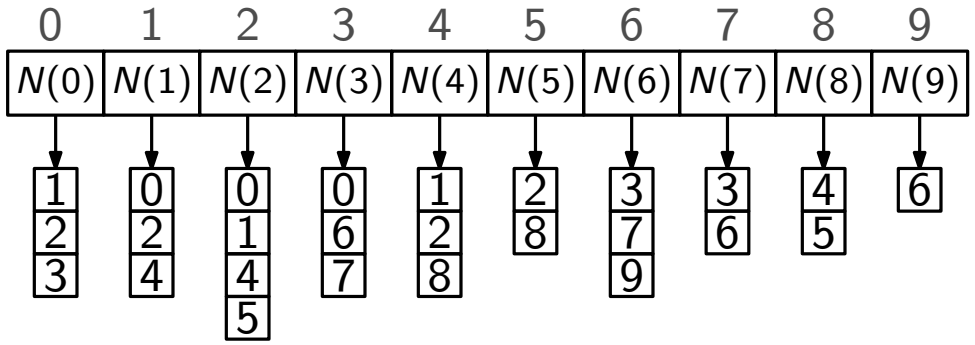


- $A[u][v] = 1 \Leftrightarrow \{u, v\} \in E$
- $N(v)$ iterieren: $\Theta(n)$
- $\Theta(n^2)$ Speicher

Fragen:

Wiederholung: Repräsentation von Graphen

Adjazenzliste



- Array A mit Nachbarschaften als Listen
- über $N(v)$ iterieren: $\Theta(\deg(v))$
- Test ob $\{u, v\} \in E$: $\Theta(\deg(v))$
- $\Theta(n + m)$ Speicher

Adjazenzmatrix

	0	1	2	3	4	5	6	7	8	9
0	0	1	1	1	0	0	0	0	0	0
1	1	0	1	0	1	0	0	0	0	0
2	1	1	0	0	1	1	0	0	0	0
3	1	0	0	0	0	0	1	1	0	0
4	0	1	1	0	0	0	0	0	1	0
5	0	0	1	0	0	0	0	0	1	0
6	0	0	0	1	0	0	0	1	0	1
7	0	0	0	1	0	0	1	0	0	0
8	0	0	0	0	1	1	0	0	0	0
9	0	0	0	0	0	0	1	0	0	0

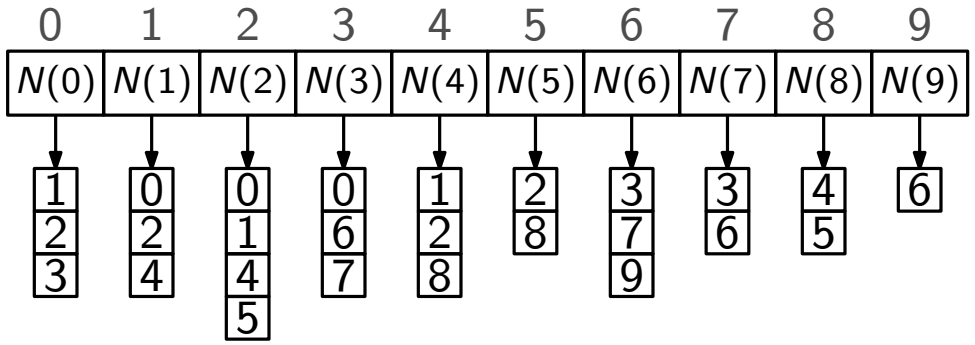
- $A[u][v] = 1 \Leftrightarrow \{u, v\} \in E$
- $N(v)$ iterieren: $\Theta(n)$
- $\Theta(n^2)$ Speicher

Fragen:

- Wie speichert man gerichtete Graphen?

Wiederholung: Repräsentation von Graphen

Adjazenzliste



- Array A mit Nachbarschaften als Listen
- über $N(v)$ iterieren: $\Theta(\deg(v))$
- Test ob $\{u, v\} \in E$: $\Theta(\deg(v))$
- $\Theta(n + m)$ Speicher

Adjazenzmatrix

	0	1	2	3	4	5	6	7	8	9
0	0	1	1	1	0	0	0	0	0	0
1	1	0	1	0	1	0	0	0	0	0
2	1	1	0	0	1	1	0	0	0	0
3	1	0	0	0	0	0	1	1	0	0
4	0	1	1	0	0	0	0	0	1	0
5	0	0	1	0	0	0	0	0	1	0
6	0	0	0	1	0	0	0	1	0	1
7	0	0	0	1	0	0	1	0	0	0
8	0	0	0	0	1	1	0	0	0	0
9	0	0	0	0	0	0	1	0	0	0

- $A[u][v] = 1 \Leftrightarrow \{u, v\} \in E$
- $N(v)$ iterieren: $\Theta(n)$
- $\Theta(n^2)$ Speicher

Fragen:

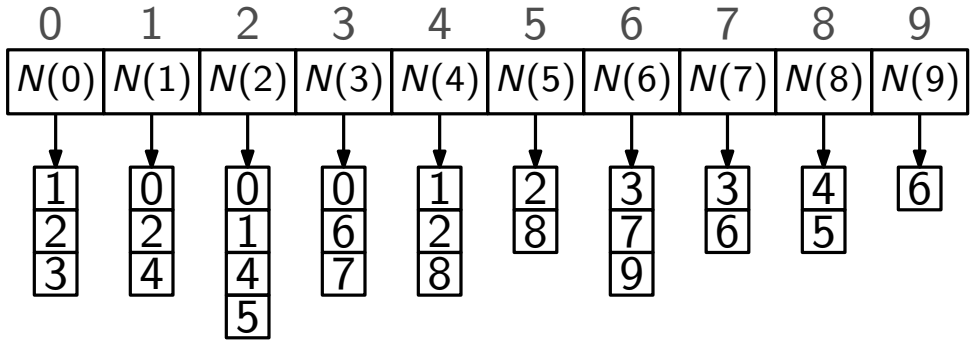
- Wie speichert man gerichtete Graphen?
- Wie speichert man andere Attribute?

Knoten löschen

Adjazenzliste

Knoten löschen

Adjazenzliste

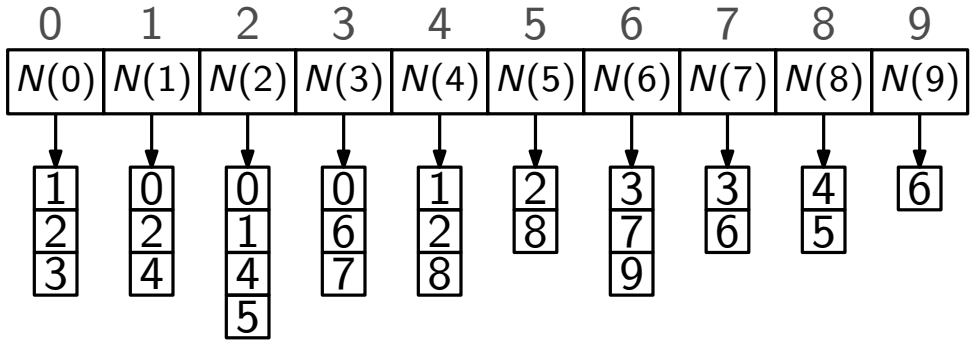


Knoten löschen

Adjazenzliste

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten

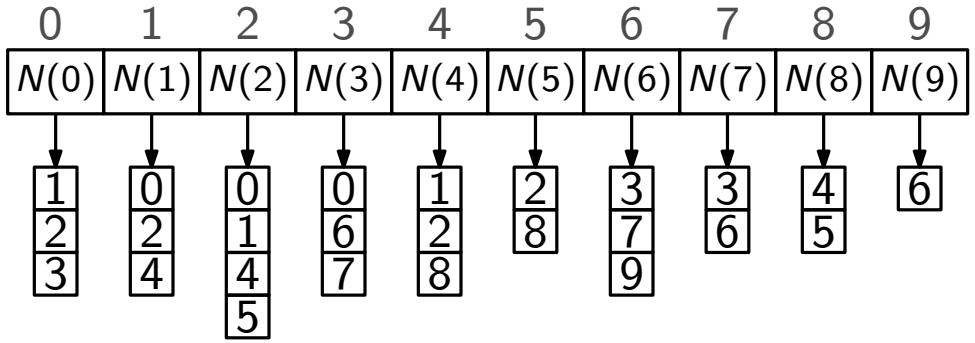


Knoten löschen

Adjazenzliste

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten



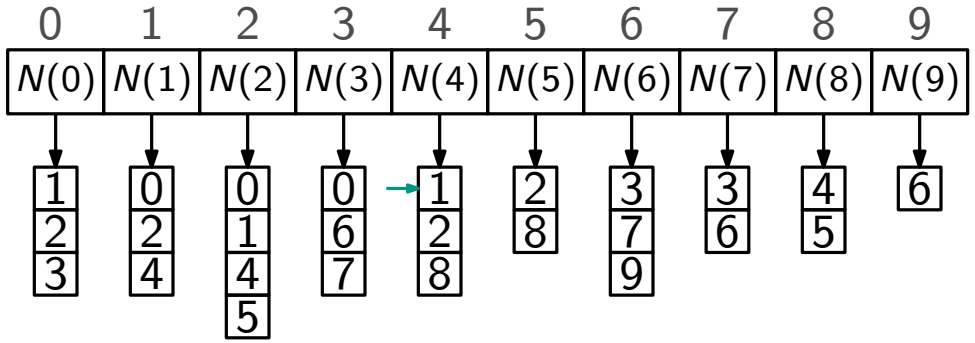
Beispiel: Lösche 4

Knoten löschen

Adjazenzliste

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten



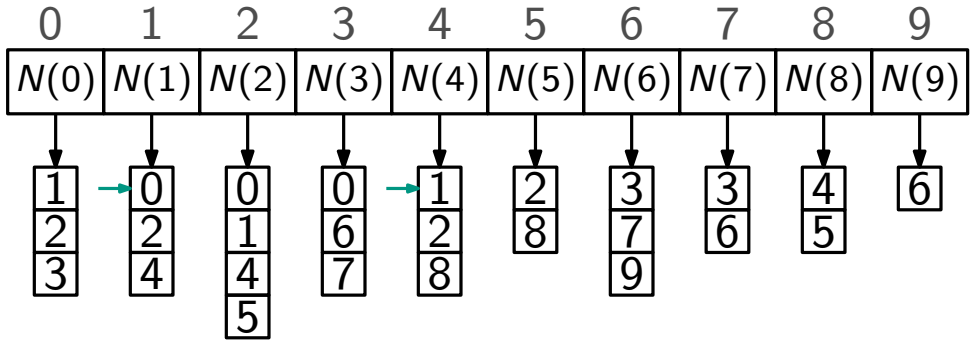
Beispiel: Lösche 4

Knoten löschen

Adjazenzliste

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten



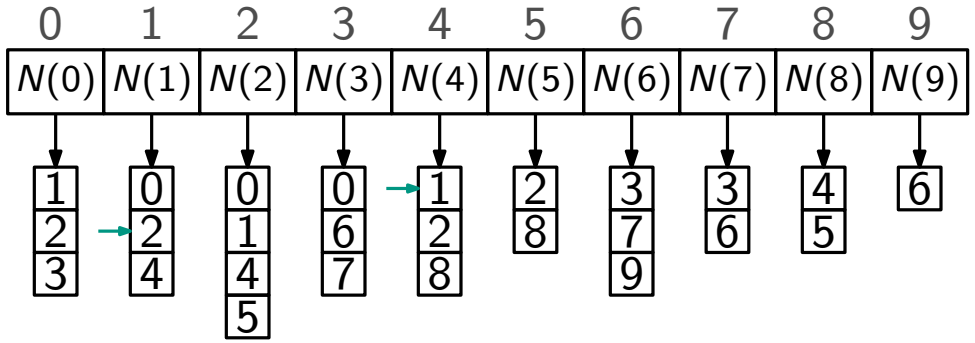
Beispiel: Lösche 4

Knoten löschen

Adjazenzliste

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten



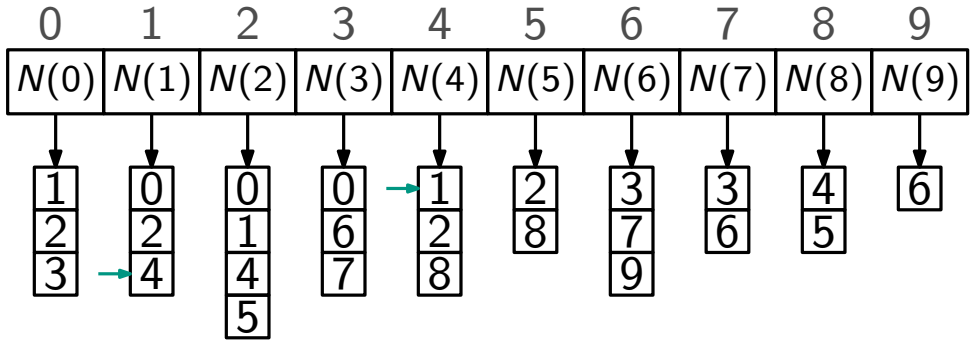
Beispiel: Lösche 4

Knoten löschen

Adjazenzliste

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten



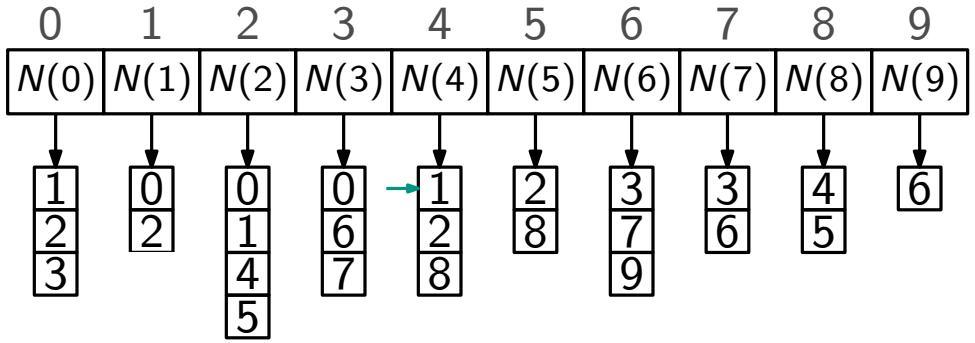
Beispiel: Lösche 4

Knoten löschen

Adjazenzliste

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten



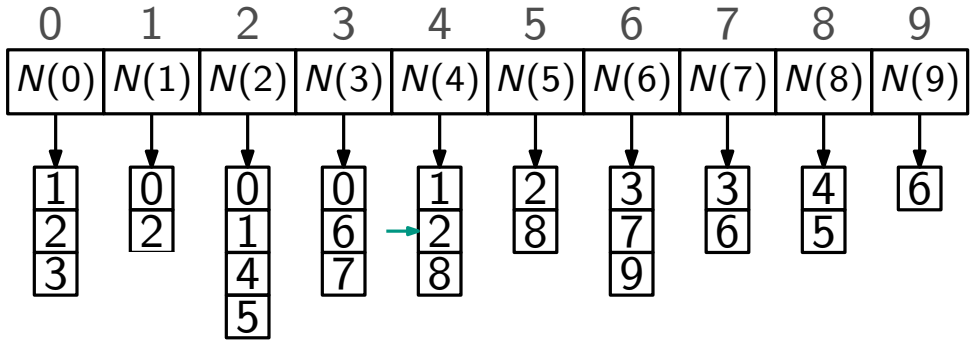
Beispiel: Lösche 4

Knoten löschen

Adjazenzliste

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten



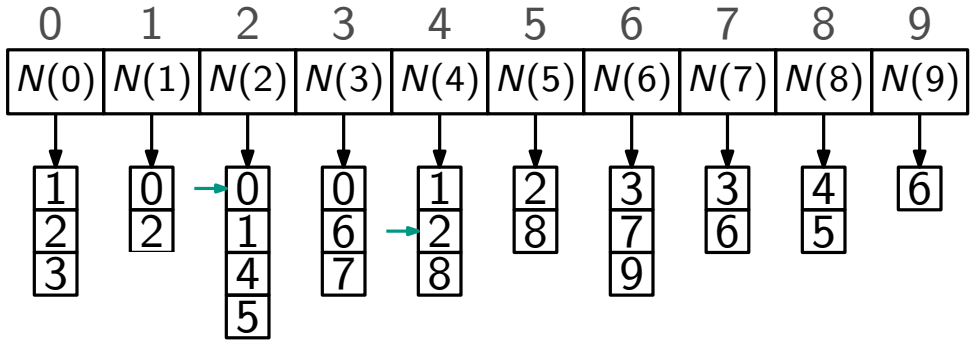
Beispiel: Lösche 4

Knoten löschen

Adjazenzliste

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten



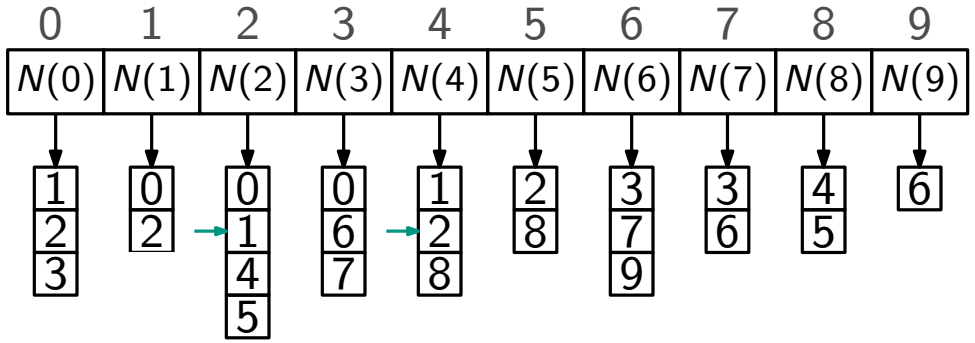
Beispiel: Lösche 4

Knoten löschen

Adjazenzliste

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten



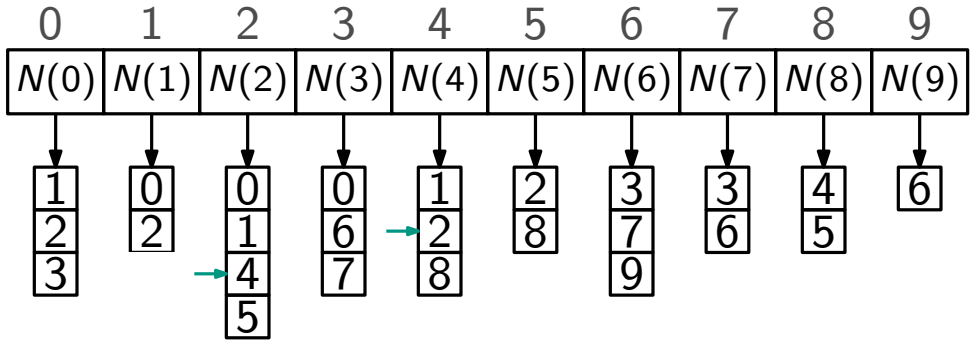
Beispiel: Lösche 4

Knoten löschen

Adjazenzliste

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten



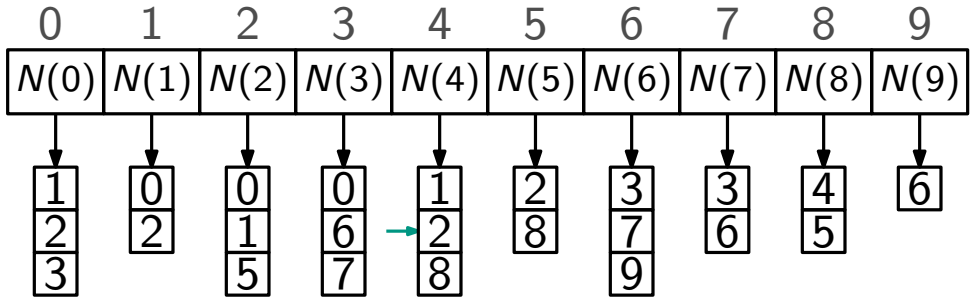
Beispiel: Lösche 4

Knoten löschen

Adjazenzliste

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten



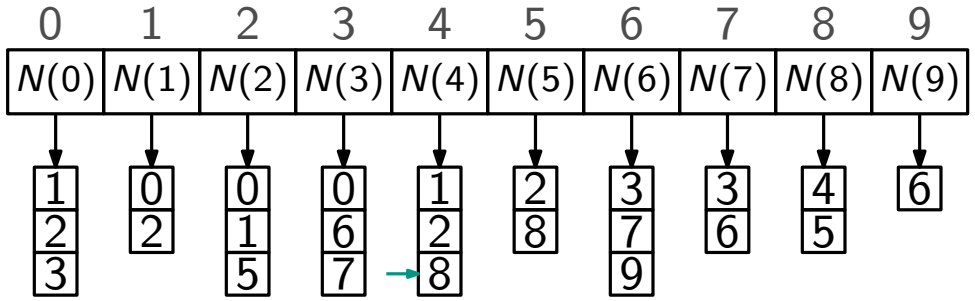
Beispiel: Lösche 4

Knoten löschen

Adjazenzliste

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten



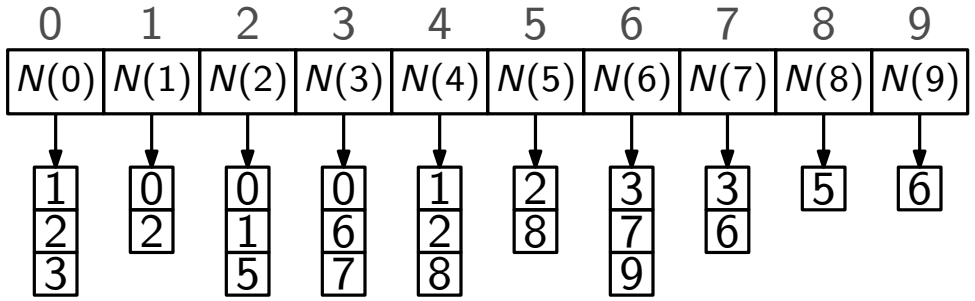
Beispiel: Lösche 4

Knoten löschen

Adjazenzliste

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten



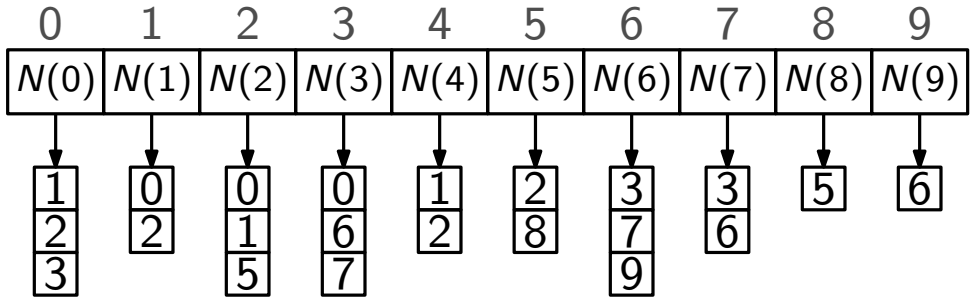
Beispiel: Lösche 4

Knoten löschen

Adjazenzliste

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten



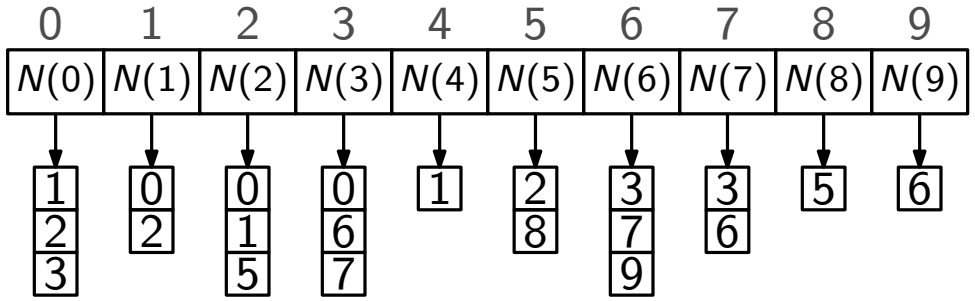
Beispiel: Lösche 4

Knoten löschen

Adjazenzliste

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten



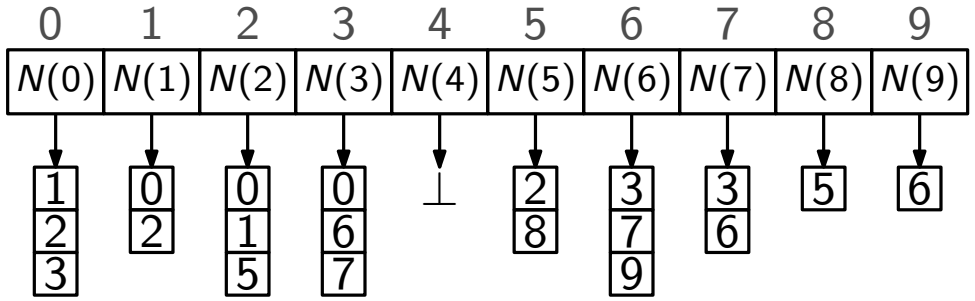
Beispiel: Lösche 4

Knoten löschen

Adjazenzliste

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten



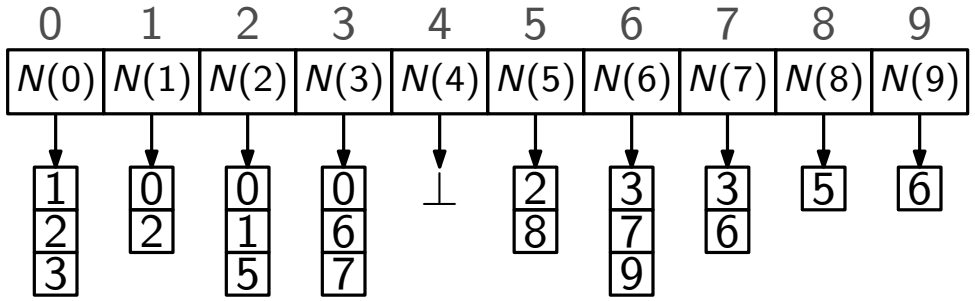
Beispiel: Lösche 4

Knoten löschen

Adjazenzliste

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten ✓
- Kanten von gelöschten Knoten ✓



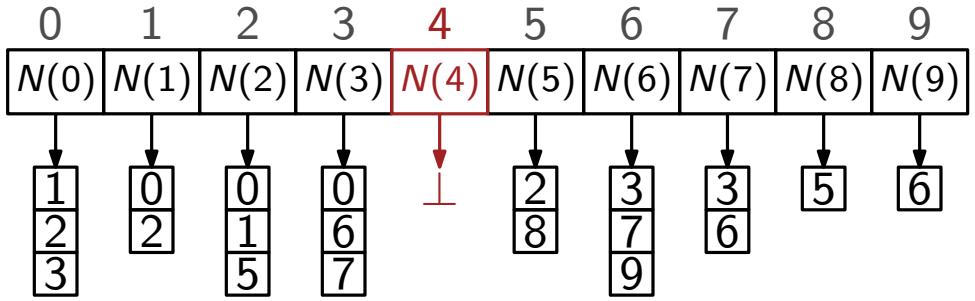
Beispiel: Lösche 4

Knoten löschen

Adjazenzliste

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten ✓
- Kanten von gelöschten Knoten ✓



Beispiel: Lösche 4

Knoten löschen

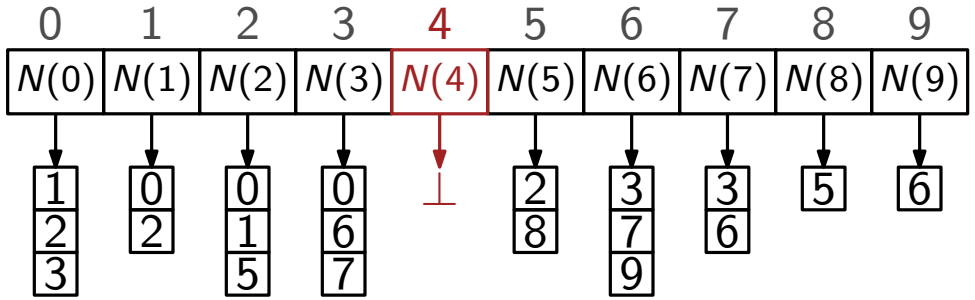
Adjazenzliste

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten

teuer ✓

✓



Beispiel: Lösche 4

Knoten löschen

Adjazenzliste

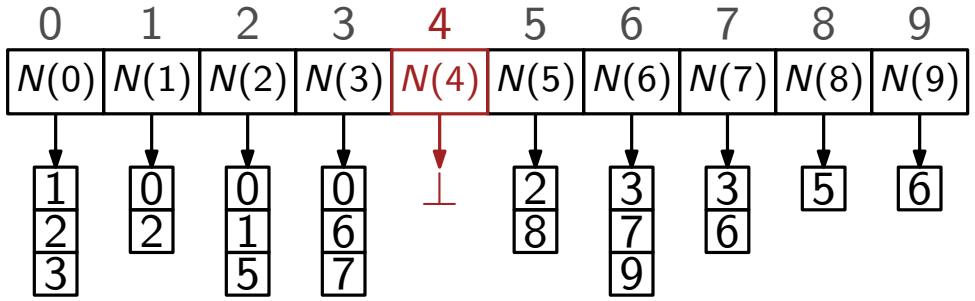
Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten

teuer ✓

✓

Besserer Ansatz



Beispiel: Lösche 4

Knoten löschen

Adjazenzliste

Naiver Ansatz: lösche

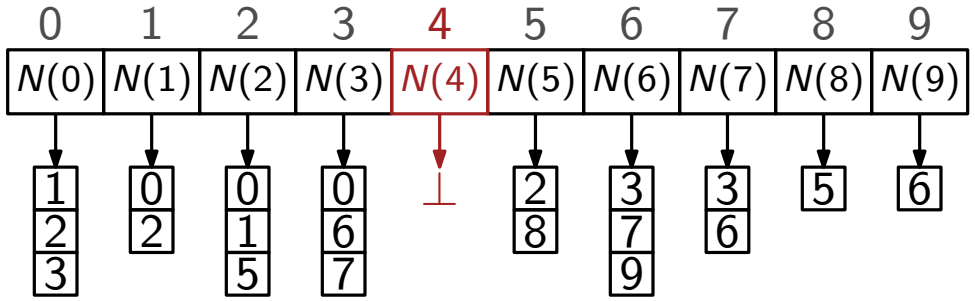
- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten

teuer ✓

✓

Besserer Ansatz

- speichere Pointer zwischen Listen



Beispiel: Lösche 4

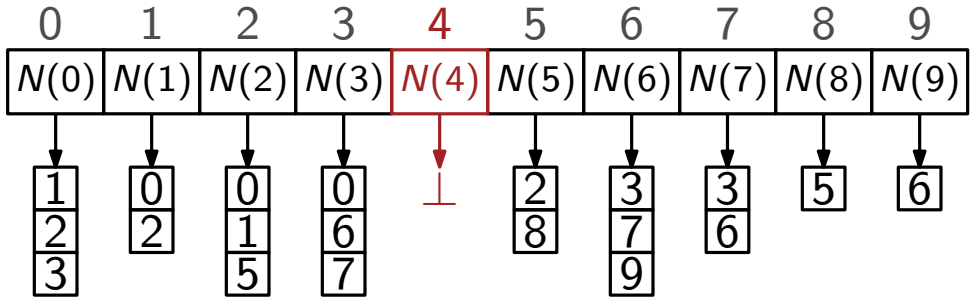
Knoten löschen

Adjazenzliste

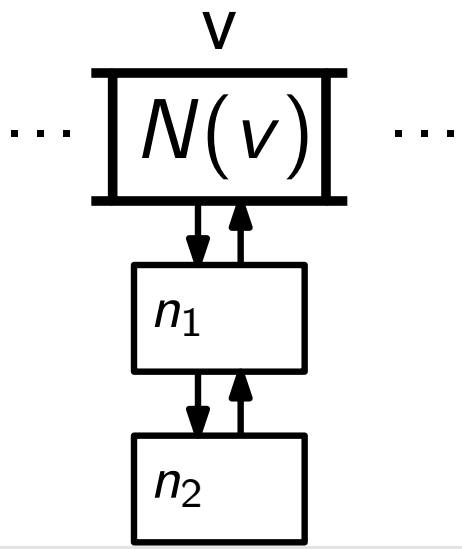
Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten

teuer ✓ ✓



Beispiel: Lösche 4



Besserer Ansatz

- speichere Pointer zwischen Listen

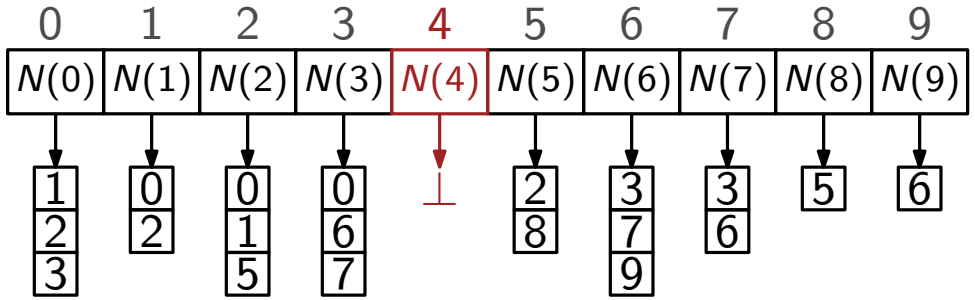
Knoten löschen

Adjazenzliste

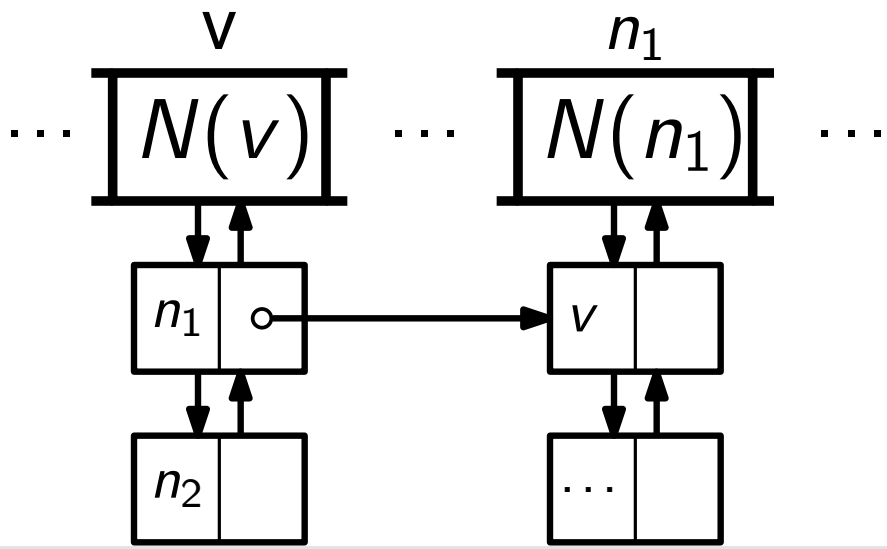
Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten

teuer ✓ ✓



Beispiel: Lösche 4



Besserer Ansatz

- speichere Pointer zwischen Listen

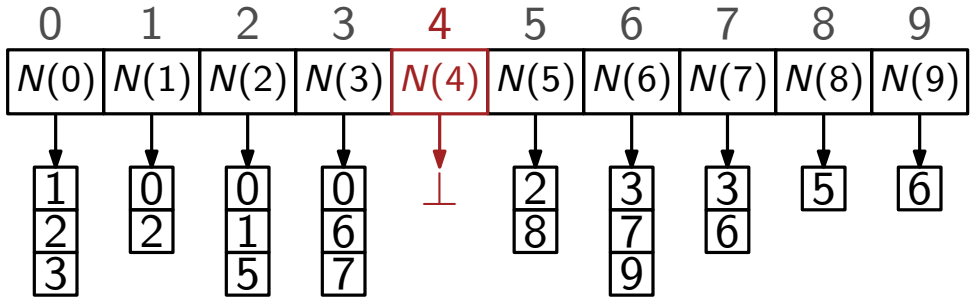
Knoten löschen

Adjazenzliste

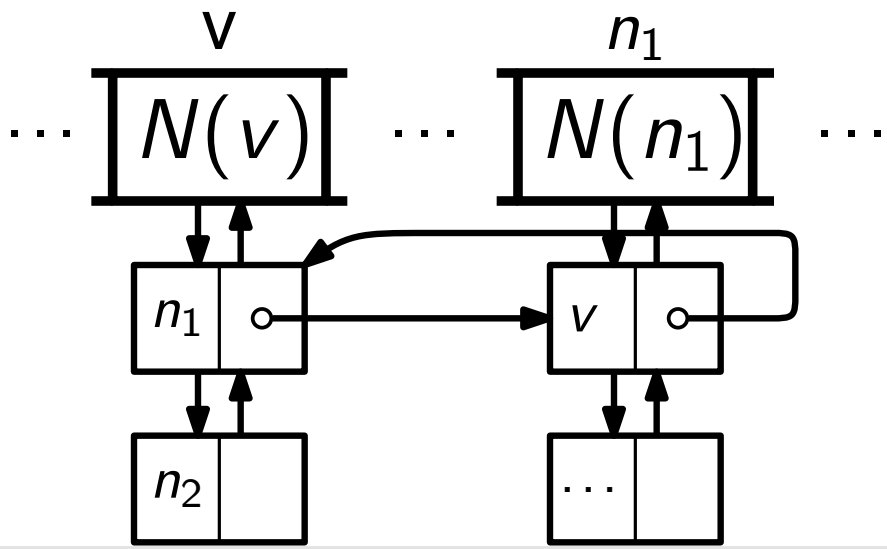
Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten

teuer ✓ ✓



Beispiel: Lösche 4



Besserer Ansatz

- speichere Pointer zwischen Listen

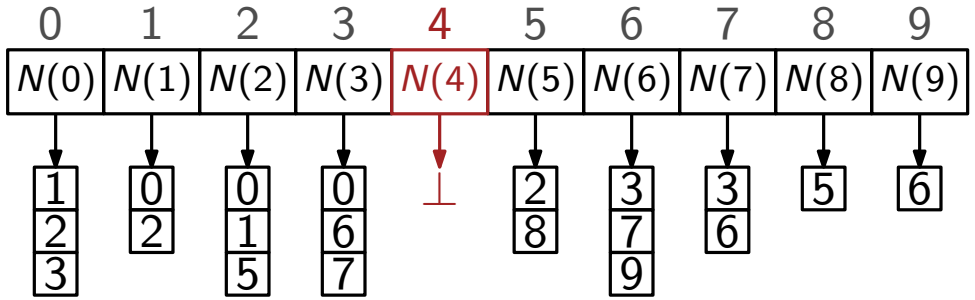
Knoten löschen

Adjazenzliste

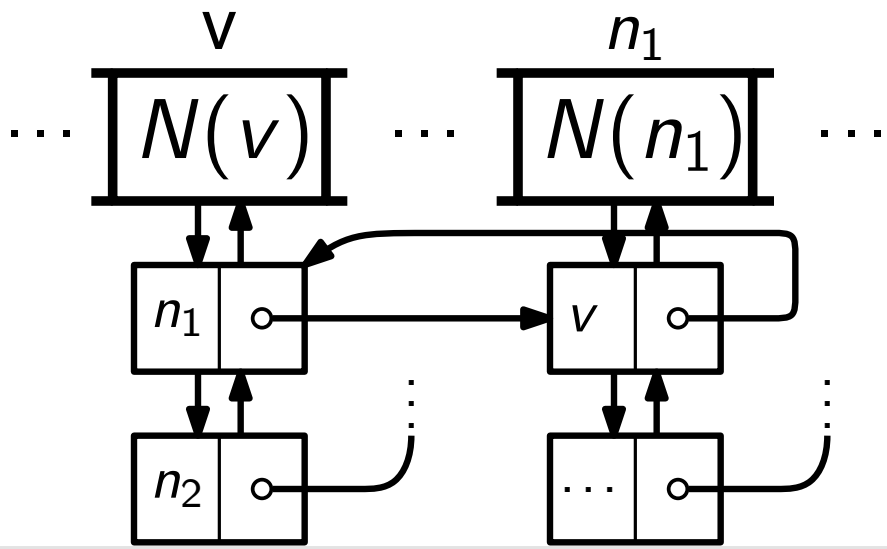
Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten

teuer ✓ ✓



Beispiel: Lösche 4



Besserer Ansatz

- speichere Pointer zwischen Listen

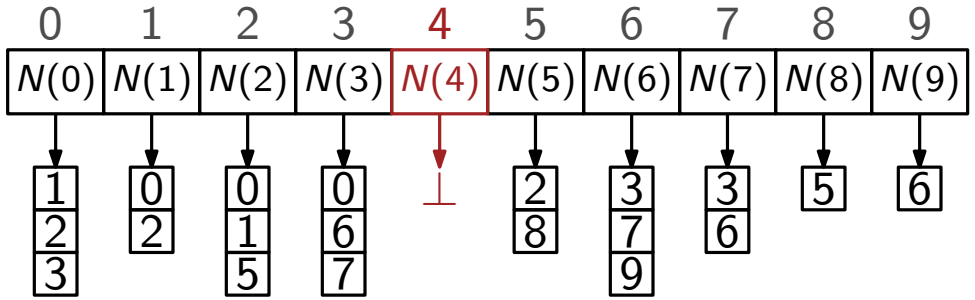
Knoten löschen

Adjazenzliste

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten

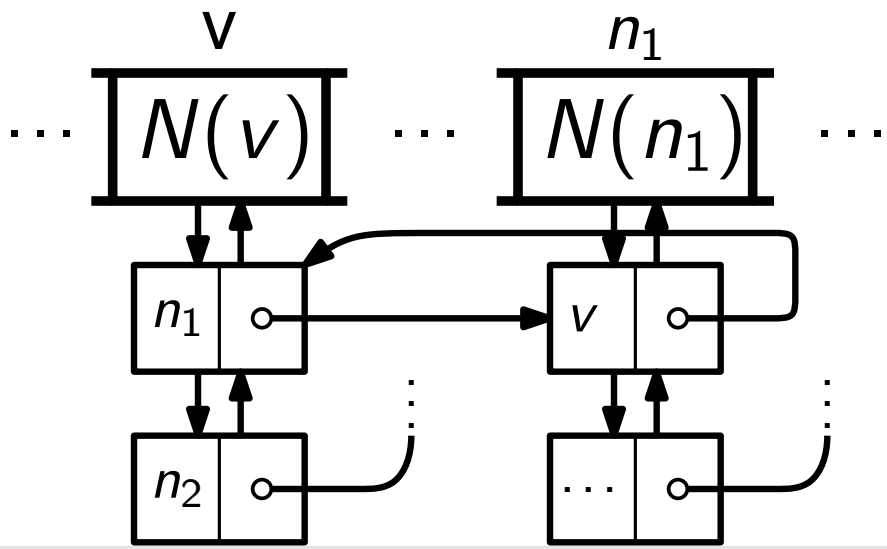
teuer ✓ ✓



Beispiel: Lösche 4

Besserer Ansatz

- speichere Pointer zwischen Listen
  - lösche inzidente Kanten in $\Theta(\deg(v))$



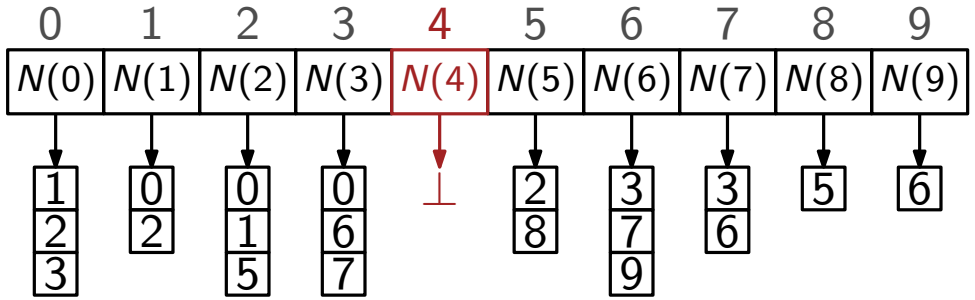
Knoten löschen

Adjazenzliste

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten

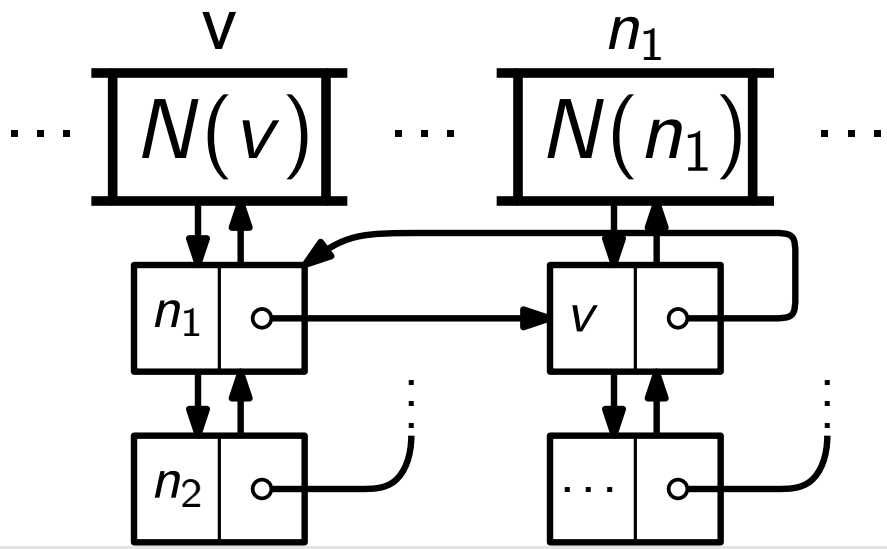
teuer ✓ ✓



Beispiel: Lösche 4

Besserer Ansatz

- speichere Pointer zwischen Listen
  - lösche inzidente Kanten in $\Theta(\deg(v))$
- Knoten Löschen: swap mit Knoten $n-1$



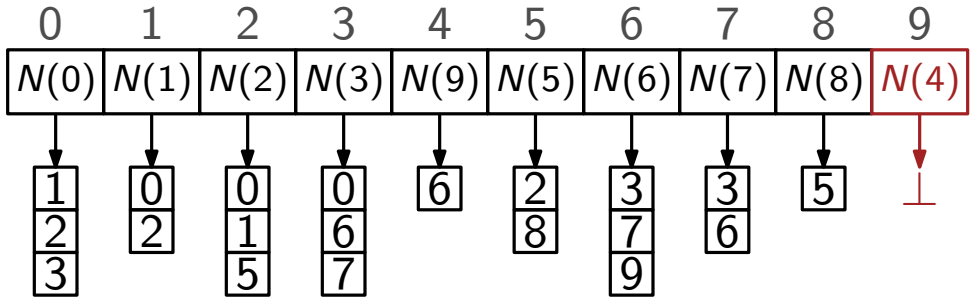
Knoten löschen

Adjazenzliste

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten

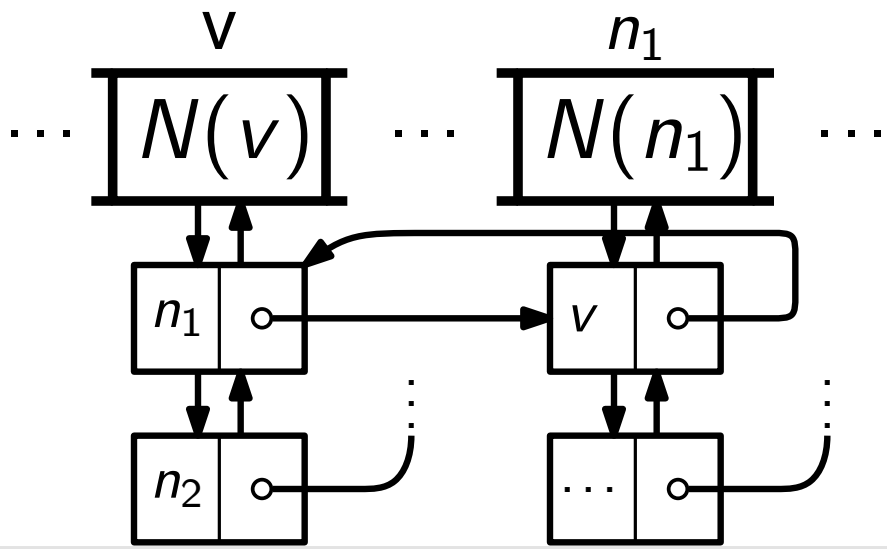
teuer ✓ ✓



Besserer Ansatz

- speichere Pointer zwischen Listen
  - lösche inzidente Kanten in $\Theta(\deg(v))$
- Knoten Löschen: swap mit Knoten $n-1$

Beispiel: Lösche 4



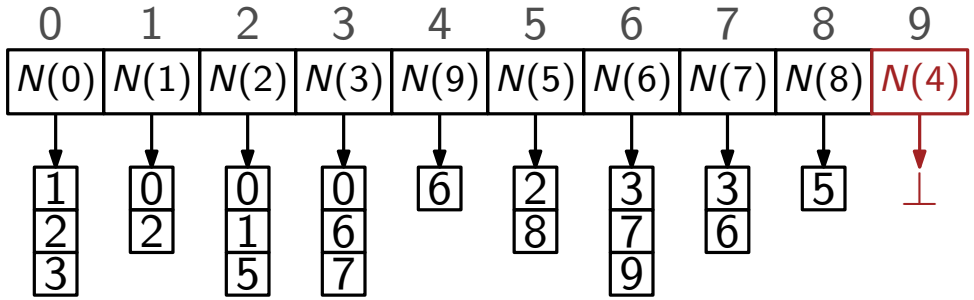
Knoten löschen

Adjazenzliste

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten

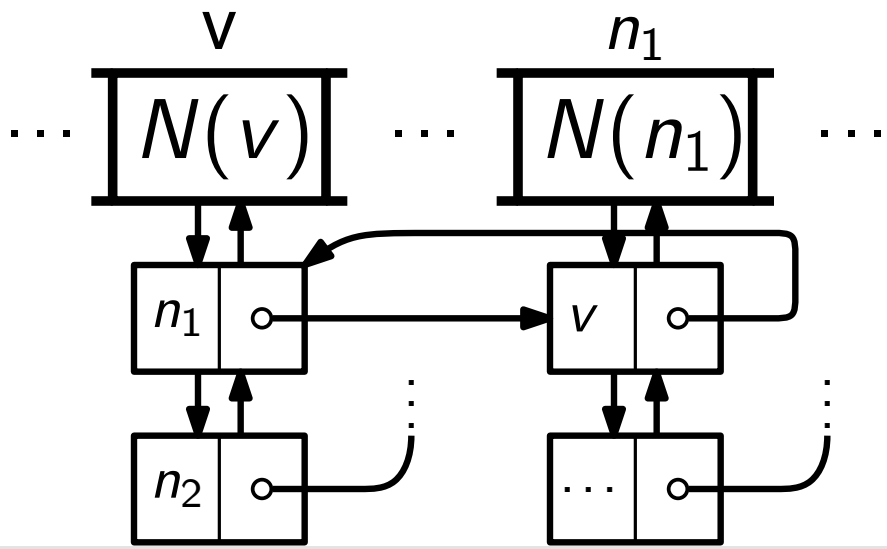
teuer ✓ ✓



Besserer Ansatz

- speichere Pointer zwischen Listen
  - lösche inzidente Kanten in $\Theta(\deg(v))$
- Knoten Löschen: swap mit Knoten $n-1$
  - mapping zw. neuen und alten Indizes
    `old_index`: $[\mathbb{N}]$, `new_index`: $[\mathbb{N}]$

Beispiel: Lösche 4



Knoten löschen

Adjazenzliste

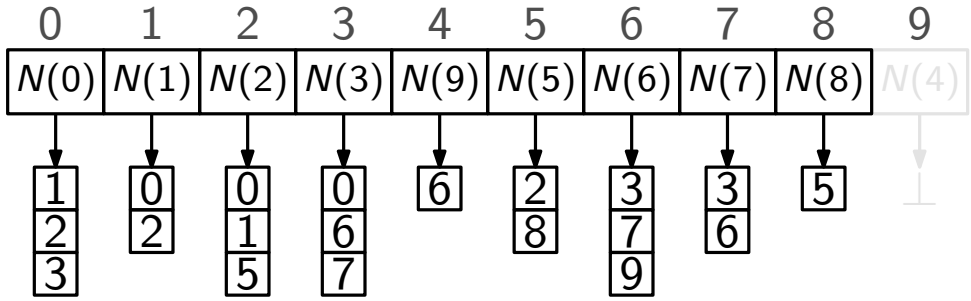
Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten

teuer ✓ ✓

Besserer Ansatz

- speichere Pointer zwischen Listen
  - lösche inzidente Kanten in $\Theta(\deg(v))$
- Knoten Löschen: swap mit Knoten $n-1$
  - mapping zw. neuen und alten Indizes
    `old_index`: $[\mathbb{N}]$, `new_index`: $[\mathbb{N}]$



Beispiel: Lösche 4

Aktualisierung Indizes:

`new_index`[9] = 4

`old_index`[4] = 9

Knoten löschen

Adjazenzliste

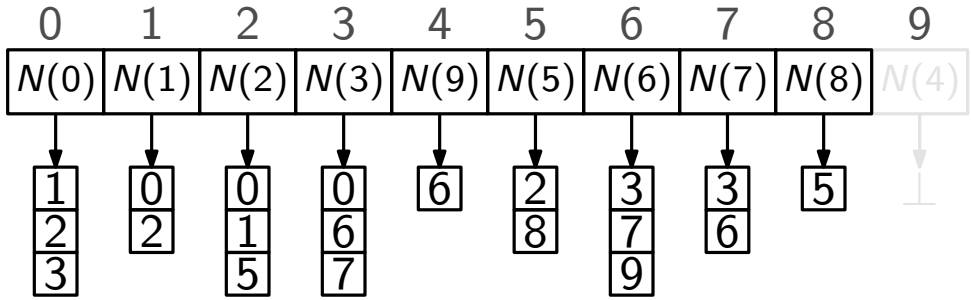
Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten

teuer ✓ ✓

Besserer Ansatz

- speichere Pointer zwischen Listen
  - lösche inzidente Kanten in $\Theta(\deg(v))$
- Knoten Löschen: swap mit Knoten $n-1$
  - mapping zw. neuen und alten Indizes
    `old_index`: $[\mathbb{N}]$, `new_index`: $[\mathbb{N}]$



Beispiel: Lösche 4

Aktualisierung Indizes:
`new_index`[9] = 4
`old_index`[4] = 9

Gesamtlaufzeit: $\Theta(\deg(v))$

Knoten löschen

Adjazenzliste

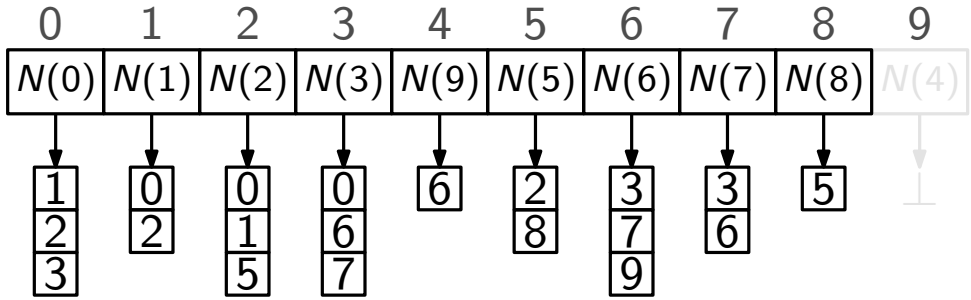
Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten

teuer ✓ ✓

Besserer Ansatz

- speichere Pointer zwischen Listen
  - lösche inzidente Kanten in $\Theta(\deg(v))$
- Knoten Löschen: swap mit Knoten $n-1$
  - mapping zw. neuen und alten Indizes
    `old_index`: $[\mathbb{N}]$, `new_index`: $[\mathbb{N}]$



Beispiel: Lösche 4

Aktualisierung Indizes:
`new_index`[9] = 4
`old_index`[4] = 9

Gesamtlaufzeit: $\Theta(\deg(v))$

Und auf gerichteten Graphen?

Knoten löschen

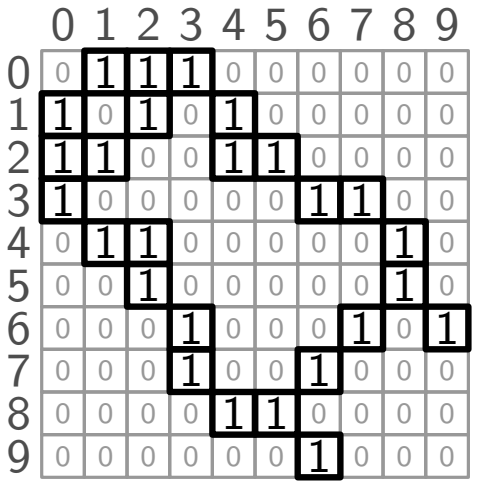
Adjazenzmatrix

- swap mit hinterstem Element
  - in beiden Dimensionen

Knoten löschen

Adjazenzmatrix

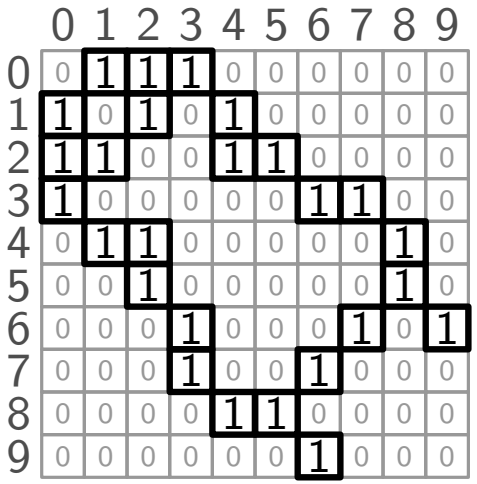
- swap mit hinterstem Element
  - in beiden Dimensionen

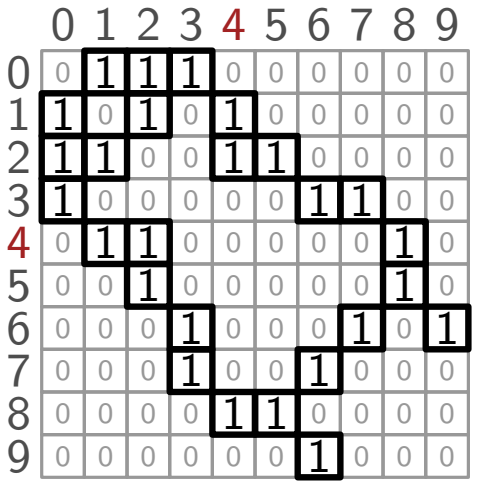


Knoten löschen

Adjazenzmatrix

- swap mit hinterstem Element
  - in beiden Dimensionen

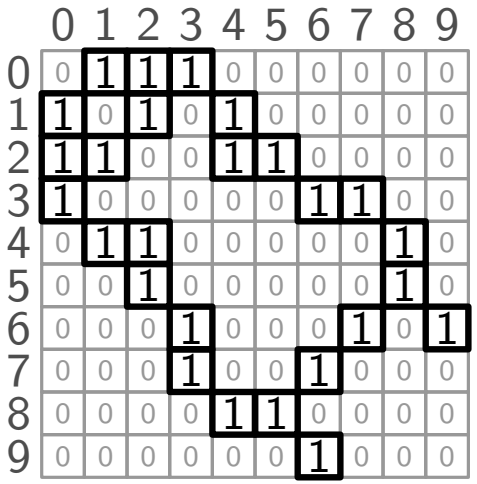


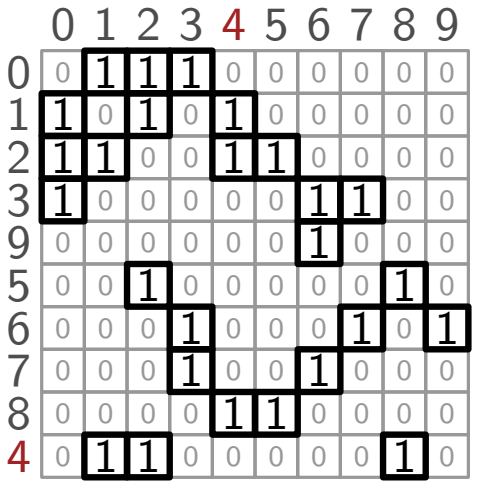


Knoten löschen

Adjazenzmatrix

- swap mit hinterstem Element
  - in beiden Dimensionen

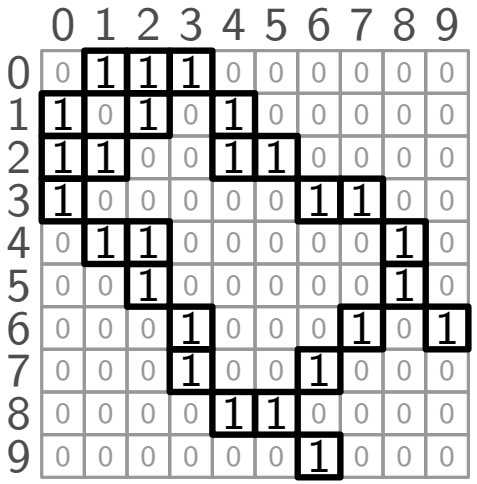


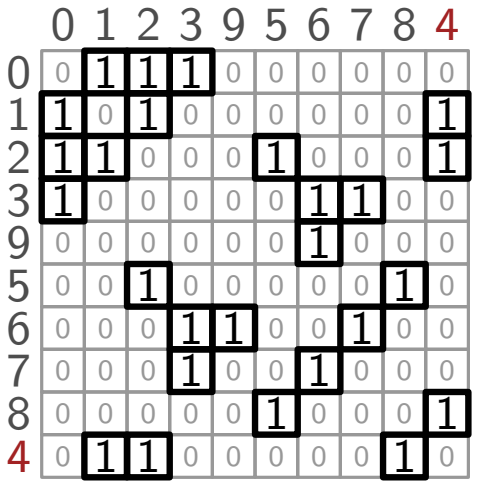


Knoten löschen

Adjazenzmatrix

- swap mit hinterstem Element
  - in beiden Dimensionen

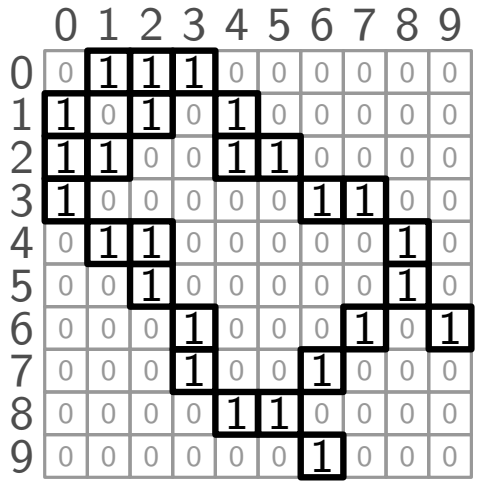


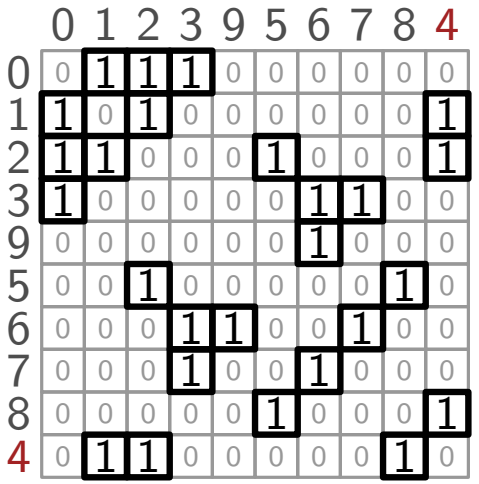


Knoten löschen

Adjazenzmatrix

- swap mit hinterstem Element
  - in beiden Dimensionen



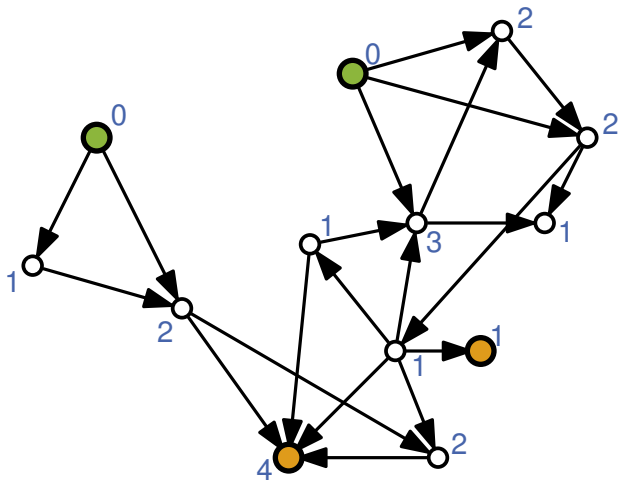


Laufzeit: $\Theta(n)$

Knoten löschen?

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Ist G DAG?



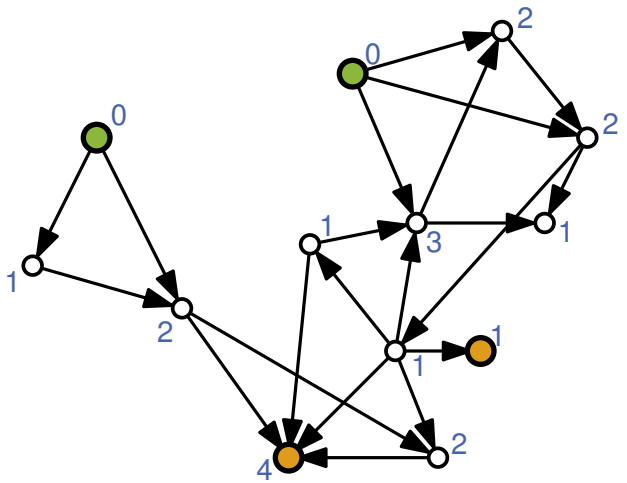
Knoten löschen?

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Ist G DAG?

Algorithmus

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q $\Theta(\deg_{\text{out}}(v))$?
  - suche neue Quellen in $N(q)$ $\Theta(\deg_{\text{out}}(v))$



$\deg_{\text{in}} : [\mathbb{N}] = [0, 0, 2, 1, \ldots]$

sources : Queue

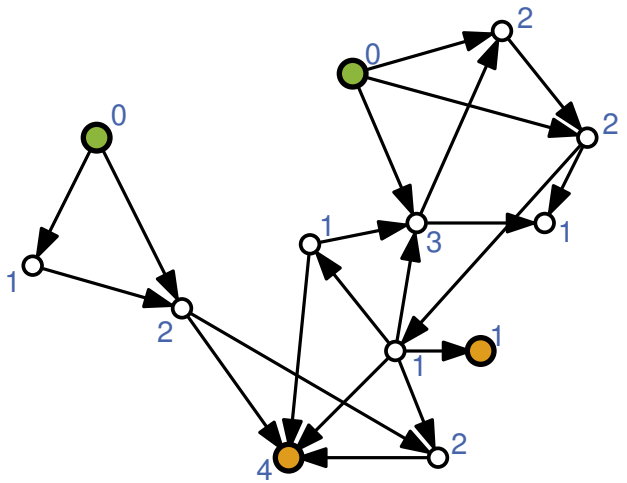
Knoten löschen?

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Ist G DAG?

Algorithmus

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q $\Theta(\deg_{\text{out}}(v))$
  - suche neue Quellen in $N(q)$ $\Theta(\deg_{\text{out}}(v))$



$\deg_{\text{in}} : [\mathbb{N}] = [0, 0, 2, 1, \ldots]$

sources : Queue

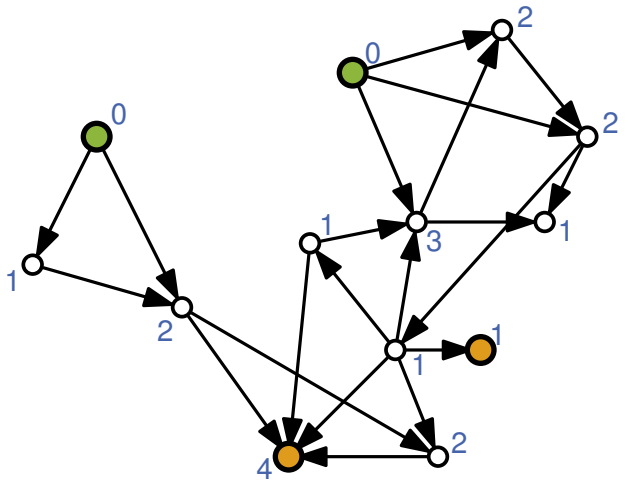
Knoten löschen?

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Ist G DAG?

Algorithmus

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q $\Theta(\deg_{\text{out}}(v))$
  - suche neue Quellen in $N(q)$ $\Theta(\deg_{\text{out}}(v))$



$\deg_{\text{in}} : [\mathbb{N}] = [0, 0, 2, 1, \ldots]$

sources : Queue

deleted : $[\mathbb{N}]$

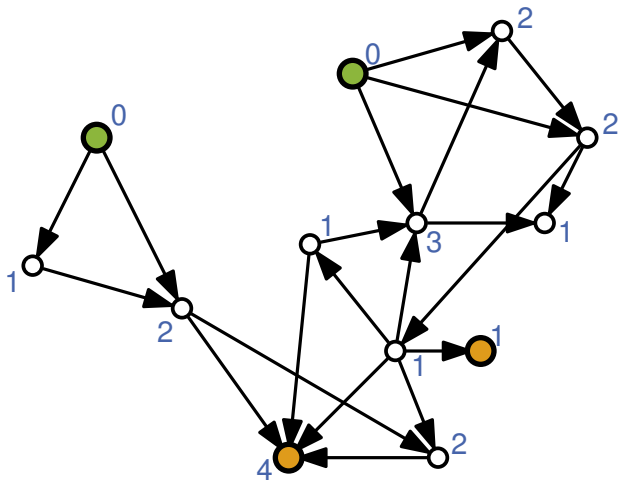
Knoten löschen?

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Ist G DAG?

Algorithmus

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q $\Theta(1)$
  - suche neue Quellen in $N(q)$ $\Theta(\deg_{\text{out}}(v))$



$\deg_{\text{in}} : [\mathbb{N}] = [0, 0, 2, 1, \ldots]$

sources : Queue

deleted : $[\mathbb{N}]$

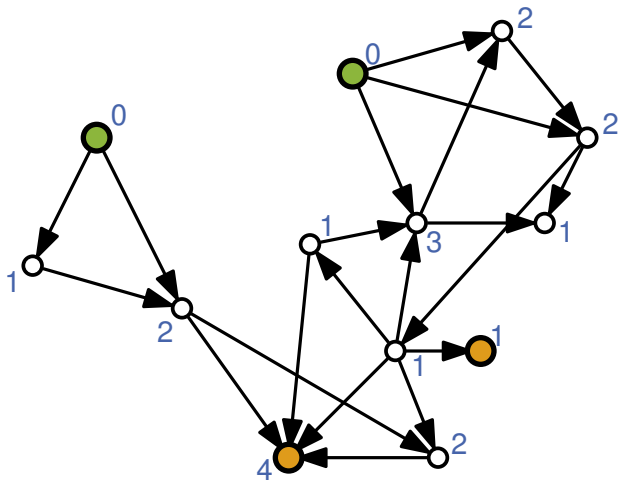
Knoten löschen?

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Ist G DAG?

Algorithmus

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q $\Theta(1)$
  - suche neue Quellen in $N(q)$ $\Theta(\deg_{\text{out}}(v))$



$\deg_{\text{in}} : [\mathbb{N}] = [0, 0, 2, 1, \ldots]$

sources : Queue

Knoten löschen?

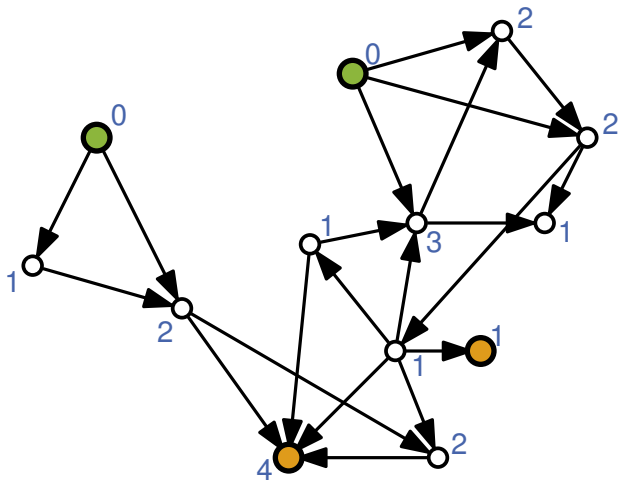
Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Ist G DAG?

Algorithmus

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q $\Theta(1)$
  - suche neue Quellen in $N(q)$ $\Theta(\deg_{\text{out}}(v))$

Gesamtlaufzeit: $\Theta(n + m)$



$\deg_{\text{in}} : [\mathbb{N}] = [0, 0, 2, 1, \ldots]$

sources : Queue

Knoten löschen?

Problemstellung

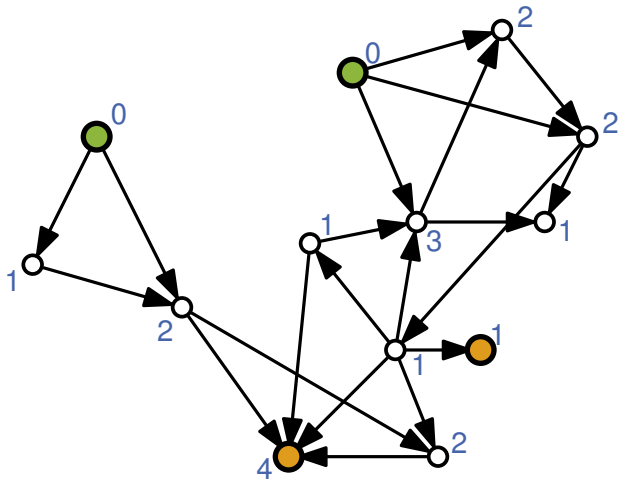
- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Ist G DAG?

Algorithmus

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q $\Theta(1)$
  - suche neue Quellen in $N(q)$ $\Theta(\deg_{\text{out}}(v))$

Gesamtlaufzeit: $\Theta(n + m)$

Wichtig



$\deg_{\text{in}} : [\mathbb{N}] = [0, 0, 2, 1, \ldots]$

sources : Queue

Knoten löschen?

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Ist G DAG?

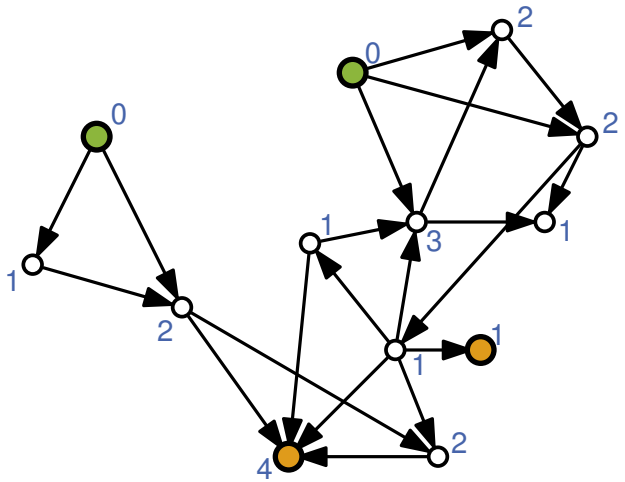
Algorithmus

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q $\Theta(1)$
  - suche neue Quellen in $N(q)$ $\Theta(\deg_{\text{out}}(v))$

Gesamtlaufzeit: $\Theta(n + m)$

Wichtig

- Knoten *löschen* oft nicht notwendig



$\deg_{\text{in}} : [\mathbb{N}] = [0, 0, 2, 1, \ldots]$

sources : Queue

Knoten löschen?

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Ist G DAG?

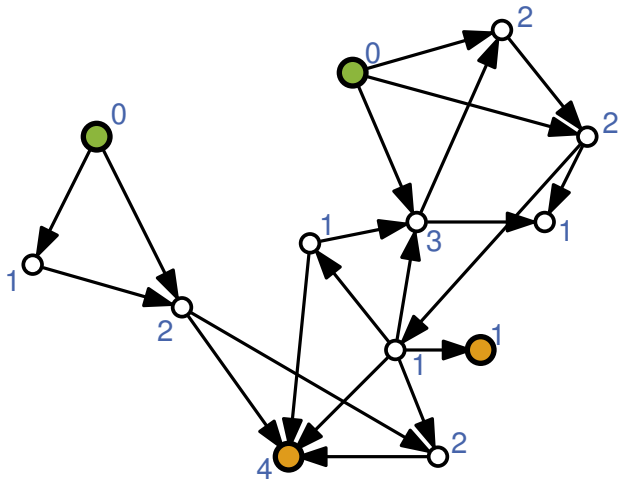
Algorithmus

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q $\Theta(1)$
  - suche neue Quellen in $N(q)$ $\Theta(\deg_{\text{out}}(v))$

Gesamtlaufzeit: $\Theta(n + m)$

Wichtig

- Knoten *löschen* oft nicht notwendig
- geschicktes Verwalten zusätzlicher Informationen hilfreich

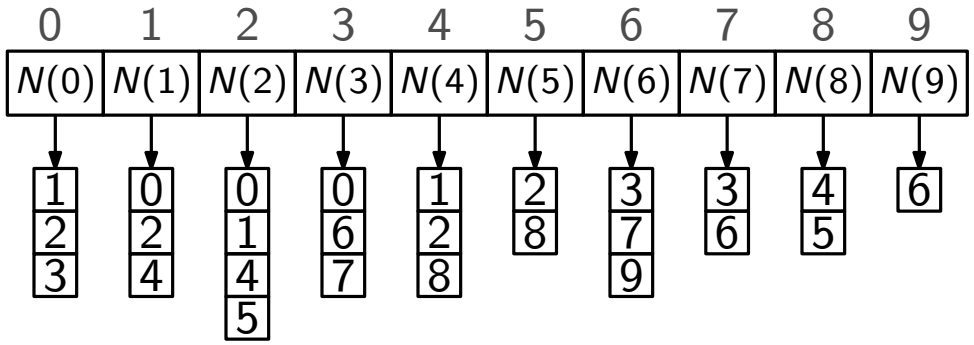


$\deg_{\text{in}} : [\mathbb{N}] = [0, 0, 2, 1, \ldots]$

sources : Queue

Repräsentation von Graphen: Varianten

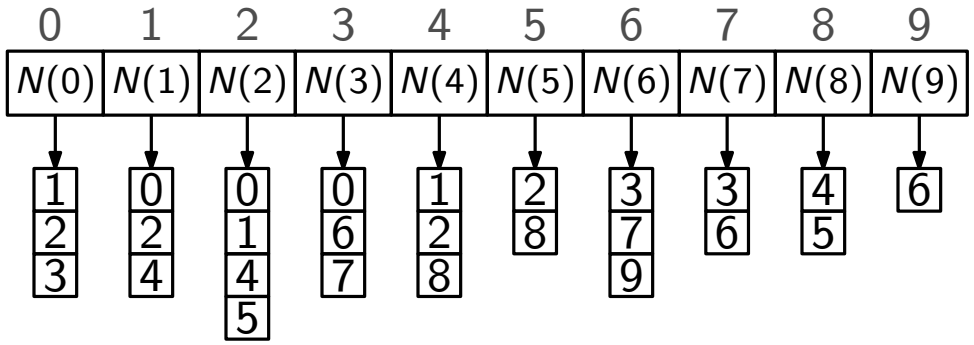
Adjazenzliste



Repräsentation von Graphen: Varianten

Adjazenzliste

Varianten

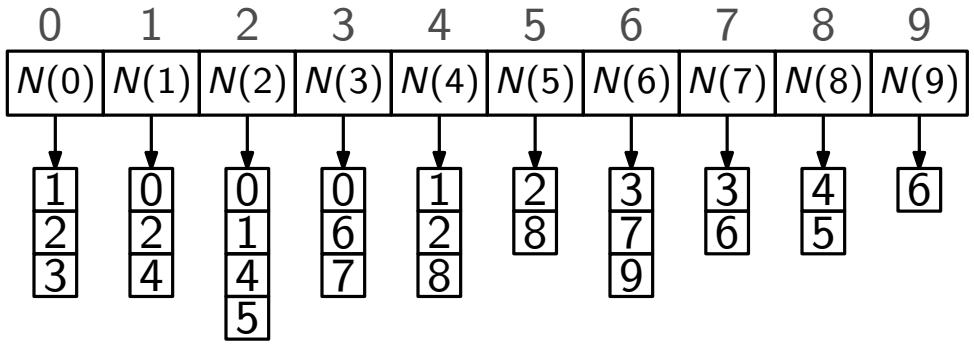


Repräsentation von Graphen: Varianten

Adjazenzliste

Varianten

- Zeiger zwischen Endpunkten von Kanten

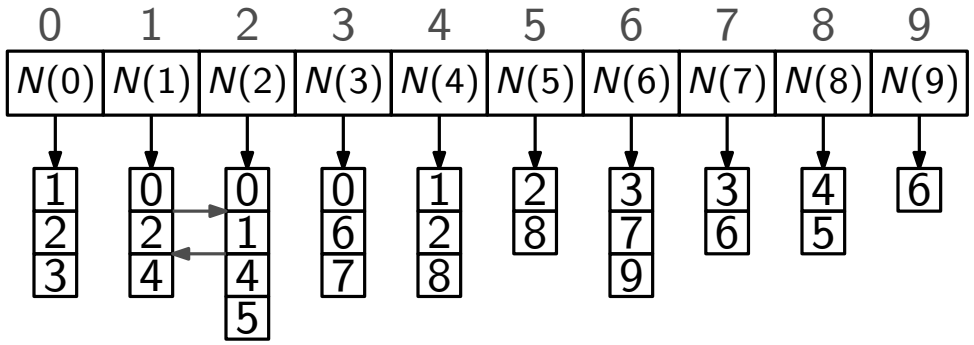


Repräsentation von Graphen: Varianten

Adjazenzliste

Varianten

- Zeiger zwischen Endpunkten von Kanten

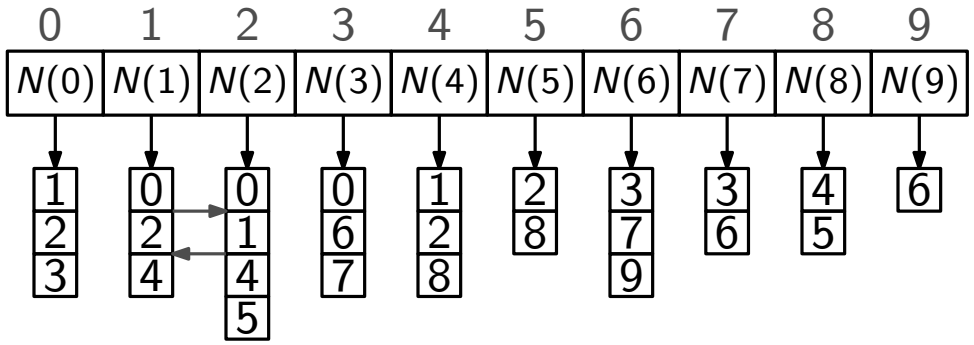


Repräsentation von Graphen: Varianten

Adjazenzliste

Varianten

- Zeiger zwischen Endpunkten von Kanten
  - nur konstanter Overhead



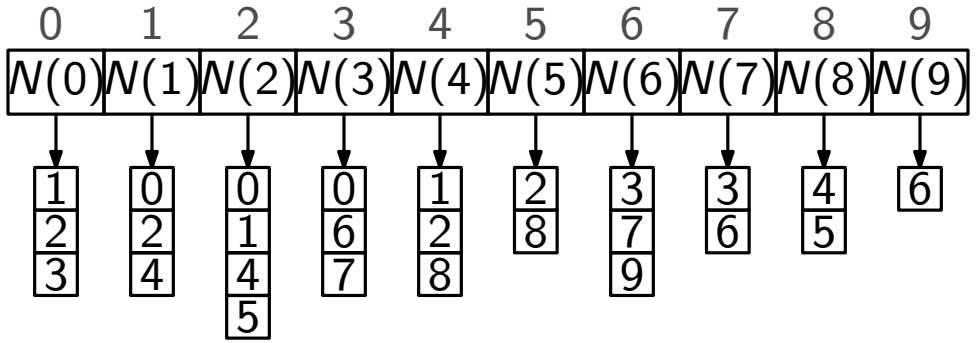
Repräsentation von Graphen: Varianten

Adjazenzliste

Varianten

- Zeiger zwischen Endpunkten von Kanten
  - nur konstanter Overhead

Adjazenzarray



Repräsentation von Graphen: Varianten

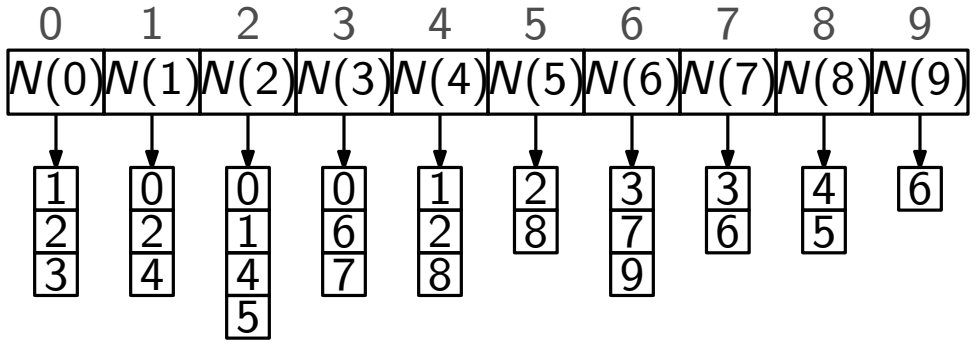
Adjazenzliste

Varianten

- Zeiger zwischen Endpunkten von Kanten
  - nur konstanter Overhead

Adjazenzarray

- Nachbarschaften als Arrays, nicht Listen



Repräsentation von Graphen: Varianten

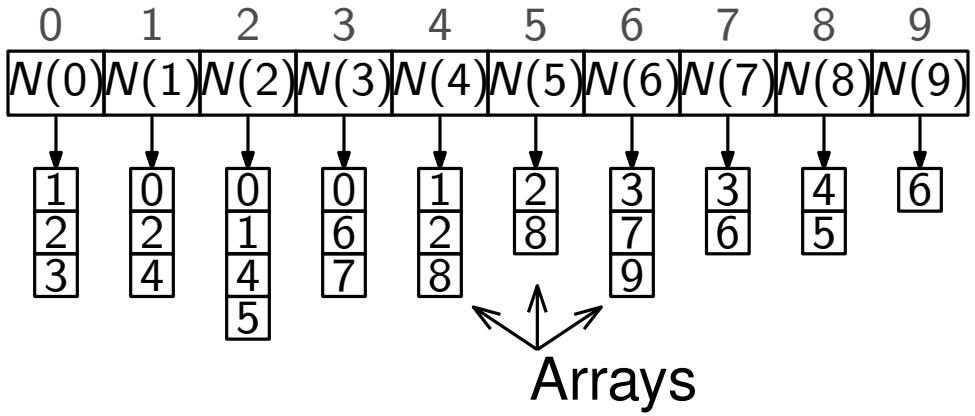
Adjazenzliste

Varianten

- Zeiger zwischen Endpunkten von Kanten
  - nur konstanter Overhead

Adjazenzarray

- Nachbarschaften als Arrays, nicht Listen



Repräsentation von Graphen: Varianten

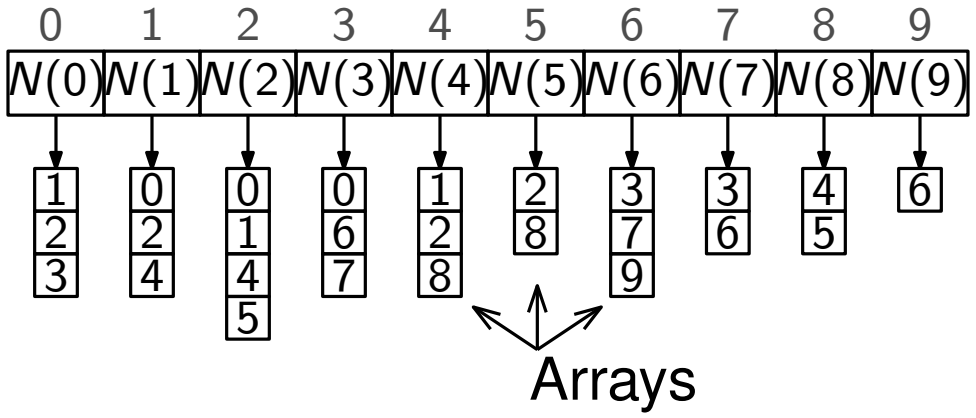
Adjazenzliste

Varianten

- Zeiger zwischen Endpunkten von Kanten
  - nur konstanter Overhead

Adjazenzarray

- Nachbarschaften als Arrays, nicht Listen
- bessere Cache Effizienz
- asymptotisch gleiche Laufzeiten



Repräsentation von Graphen: Varianten

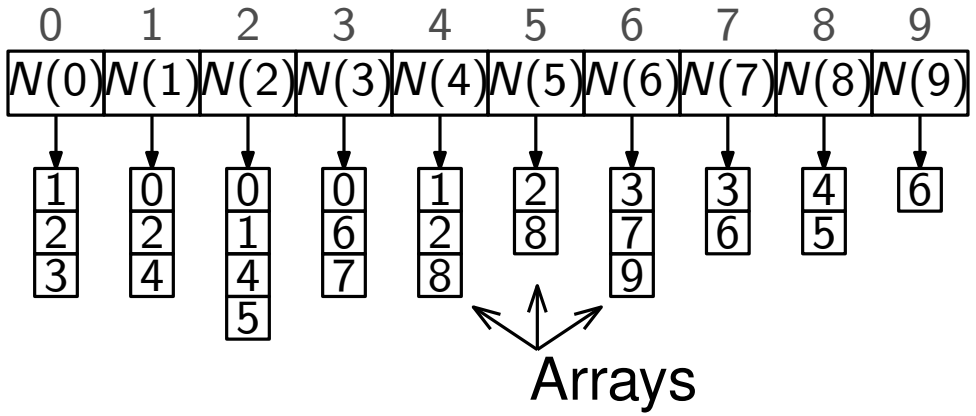
Adjazenzliste

Varianten

- Zeiger zwischen Endpunkten von Kanten
  - nur konstanter Overhead

Adjazenzarray

- Nachbarschaften als Arrays, nicht Listen
- bessere Cache Effizienz
- asymptotisch gleiche Laufzeiten
- Optimierung: sortierte Arrays für binäre Suche



Repräsentation von Graphen: Varianten

Adjazenzliste

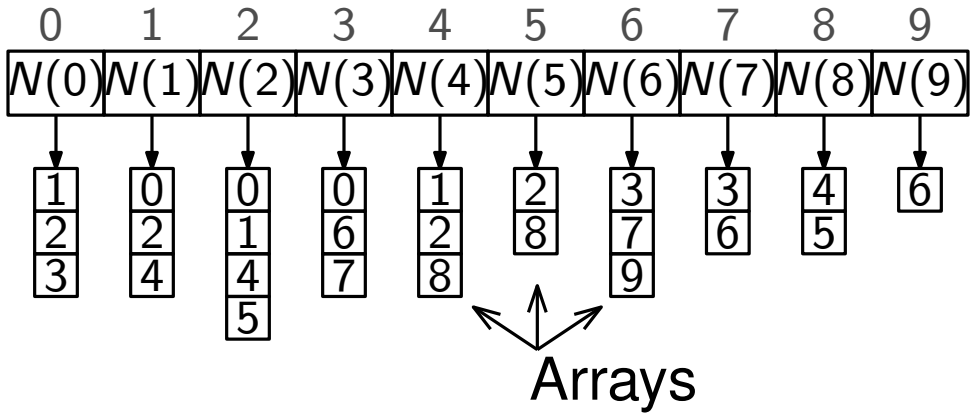
Varianten

- Zeiger zwischen Endpunkten von Kanten
  - nur konstanter Overhead

Adjazenzarray

- Nachbarschaften als Arrays, nicht Listen
- bessere Cache Effizienz
- asymptotisch gleiche Laufzeiten
- Optimierung: sortierte Arrays für binäre Suche

Sonstiges



Repräsentation von Graphen: Varianten

Adjazenzliste

Varianten

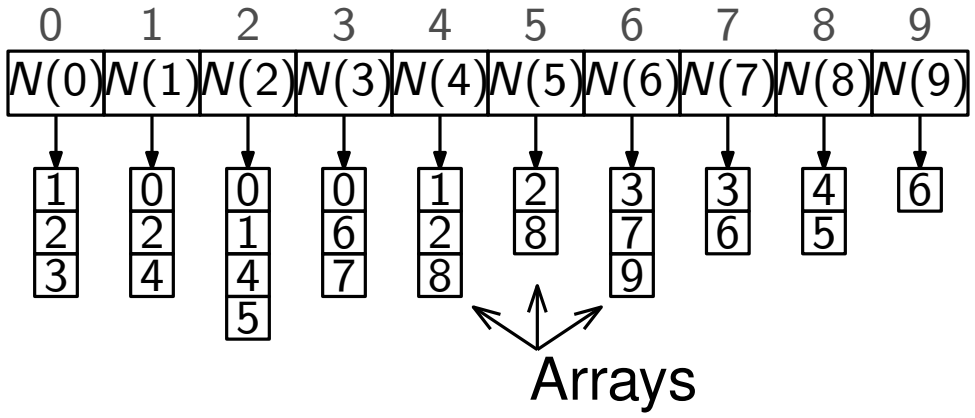
- Zeiger zwischen Endpunkten von Kanten
  - nur konstanter Overhead

Adjazenzarray

- Nachbarschaften als Arrays, nicht Listen
- bessere Cache Effizienz
- asymptotisch gleiche Laufzeiten
- Optimierung: sortierte Arrays für binäre Suche

Sonstiges

- Nachbarschaft als Bit-Vektoren



Repräsentation von Graphen: Varianten

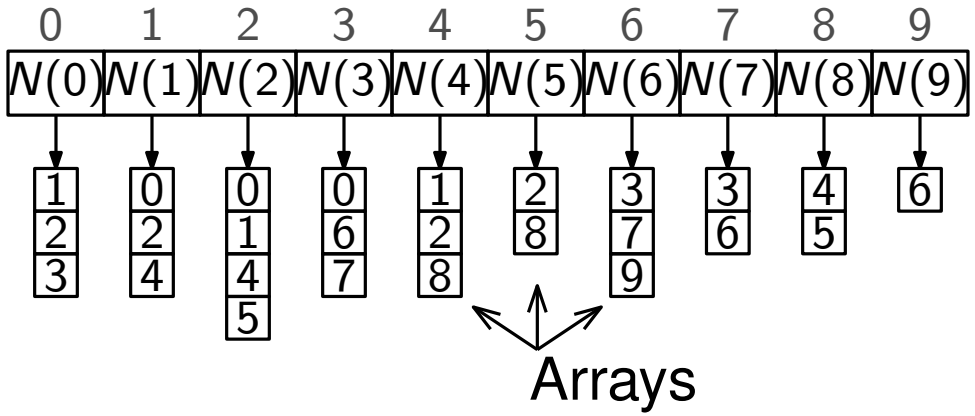
Adjazenzliste

Varianten

- Zeiger zwischen Endpunkten von Kanten
  - nur konstanter Overhead

Adjazenzarray

- Nachbarschaften als Arrays, nicht Listen
- bessere Cache Effizienz
- asymptotisch gleiche Laufzeiten
- Optimierung: sortierte Arrays für binäre Suche



Sonstiges

- Nachbarschaft als Bit-Vektoren

Repräsentation von Graphen: Varianten

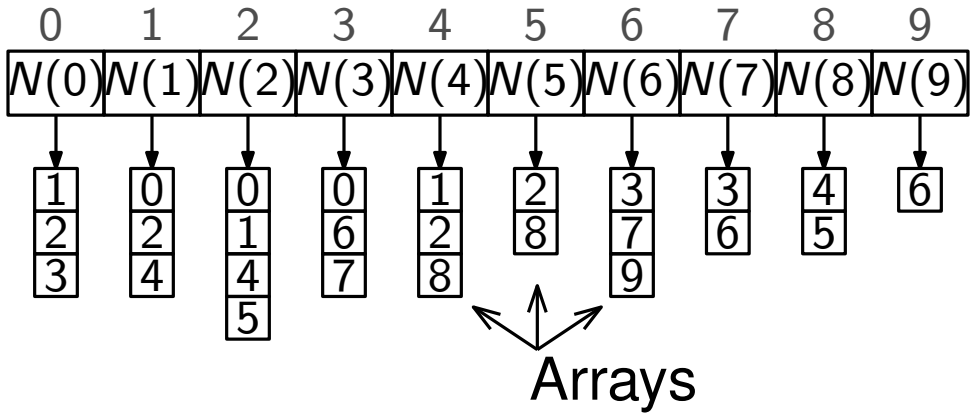
Adjazenzliste

Varianten

- Zeiger zwischen Endpunkten von Kanten
  - nur konstanter Overhead

Adjazenzarray

- Nachbarschaften als Arrays, nicht Listen
- bessere Cache Effizienz
- asymptotisch gleiche Laufzeiten
- Optimierung: sortierte Arrays für binäre Suche



Sonstiges

- Nachbarschaft als Bit-Vektoren
  - Schnelle Mengenoperationen (z.B. $N(v) \cap N(w)$)