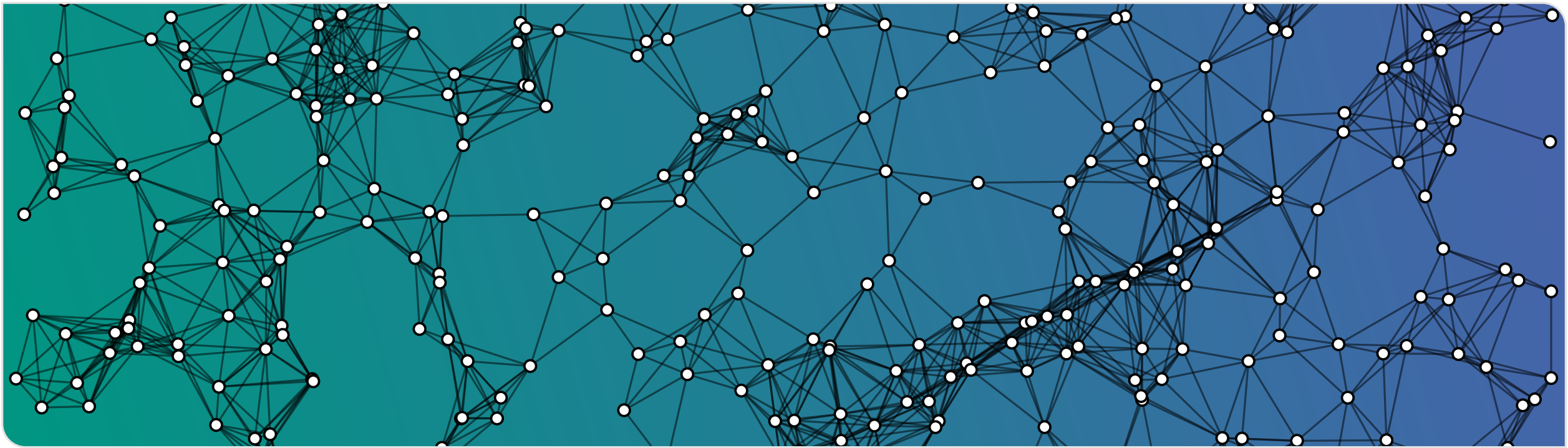
Karlsruher Institut für Technologie

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

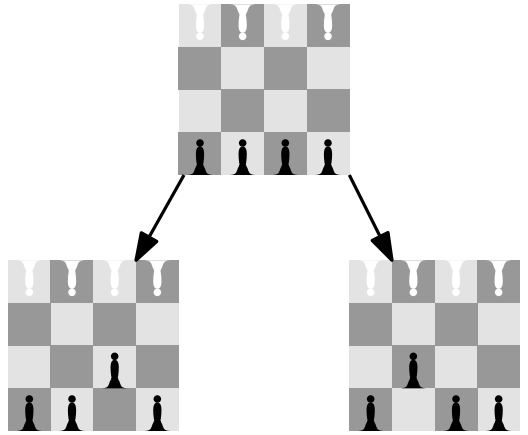
August 2022

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	1		

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

Graphen

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



Grundlagen und Notation

Graph

$G = (V, E)$

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

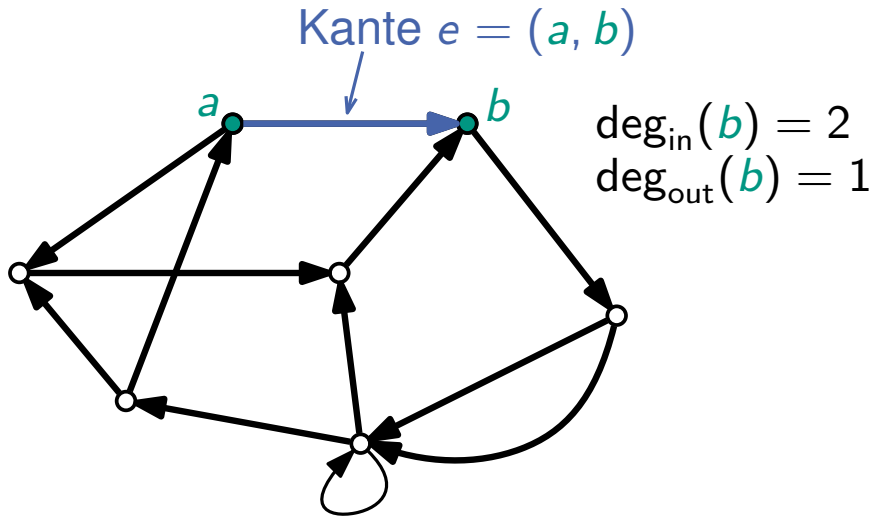
Graph (gerichtet)

$G = (V, E)$

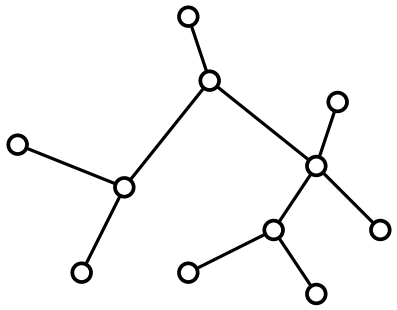
- V: endliche Menge
- E: Teilmenge von $V \times V$

Sonstiges

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



Besondere Graphen



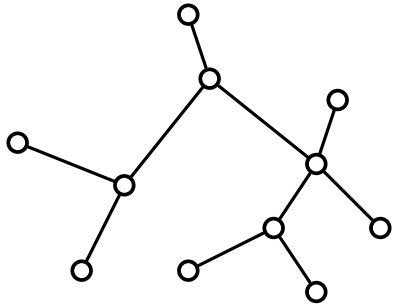
Bäume

Charakterisierung:

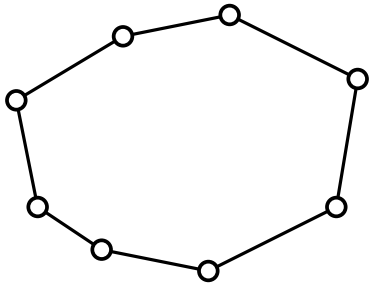
- kreisfrei
- zusammenhängend
- $m = n - 1$

} Zwei Eigenschaften implizieren dritte

Besondere Graphen



Bäume

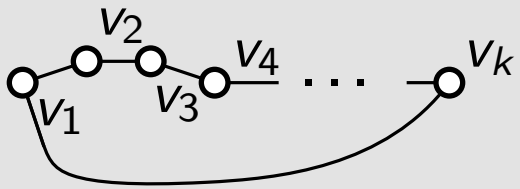


Kreise C_n

Charakterisierung:

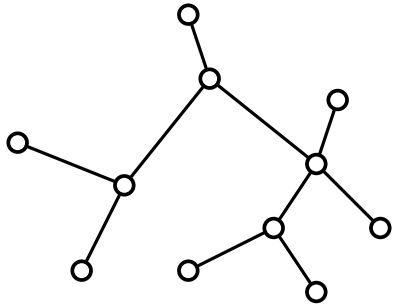
- zusammenhängend
- $\deg(v) = 2$ f.a. $v \in V$

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

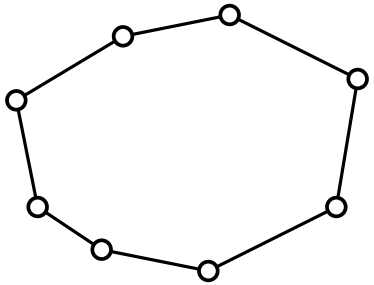


- für $2 \leq i < k$: $\{v_i, v_k\} \notin E$
- $\{v_k, v_1\} \in E$ falls $k = n$

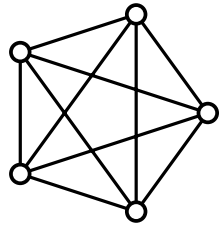
Besondere Graphen



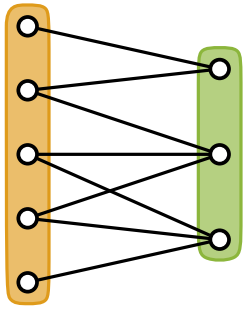
Bäume



Kreise

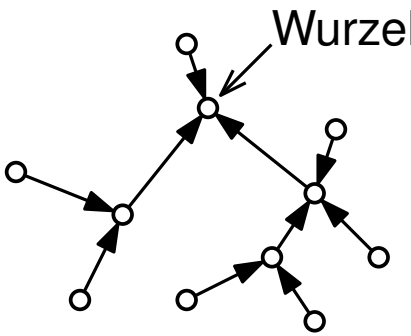


vollst. Graph K_n

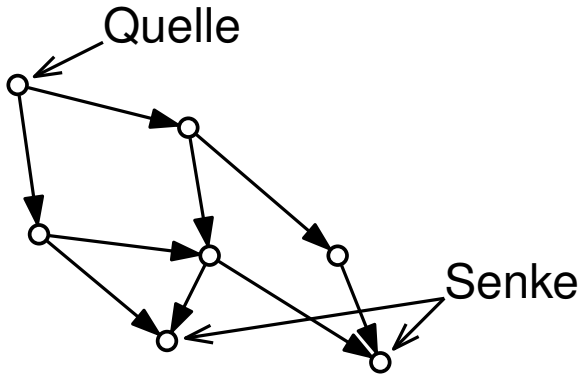


bipartiter Graph

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



In-tree / zur Wurzel ger. Baum



directed acyclic graph (DAG)

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

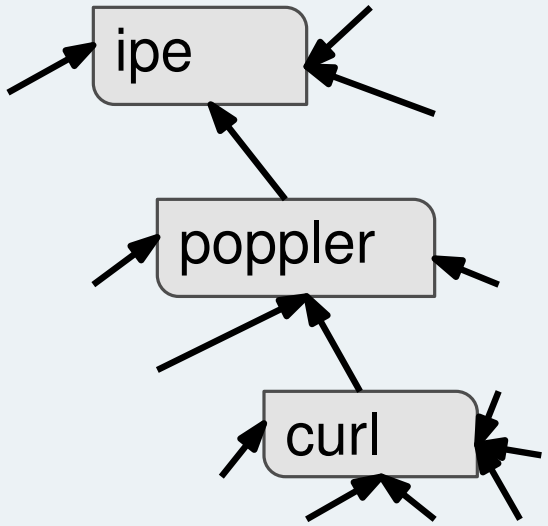
Architecture:	x86_64
Repository:	Core
Split Packages:	libcurl-compat, libcurl-gnutls
Description:	An URL retrieval utility and library
Upstream URL:	https://curl.haxx.se
License(s):	MIT
Provides:	libcurl.so=4-64
Maintainers:	Christian Hesse
Package Size:	1.1 MB
Installed Size:	1.8 MB
Last Packager:	Christian 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)*



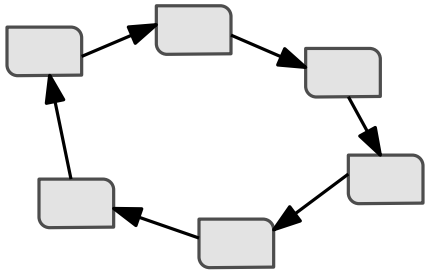
Problem: Dependencies

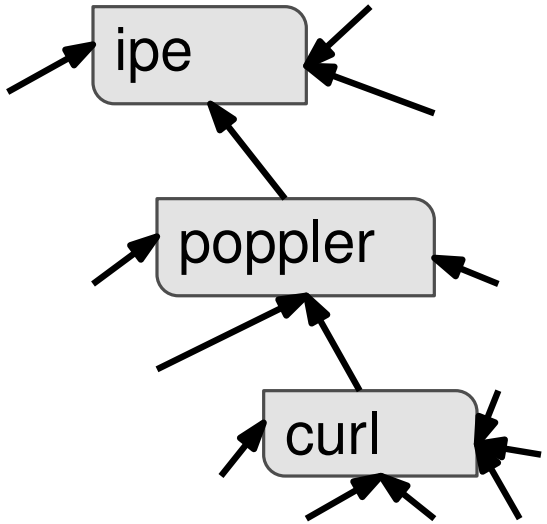
Modellierung als Graph

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

Frage

Gibt es zyklische Abhängigkeiten?





Cyclic Dependency

Problemstellung

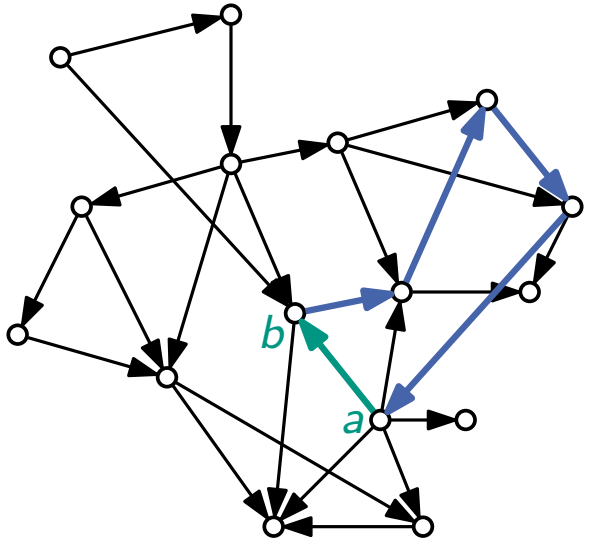
- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

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

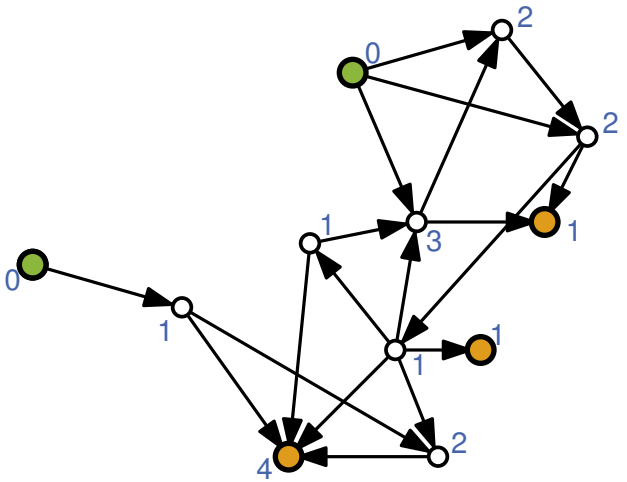
- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- 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 $N(q)$ $\Theta(\deg_{\text{out}}(v))$

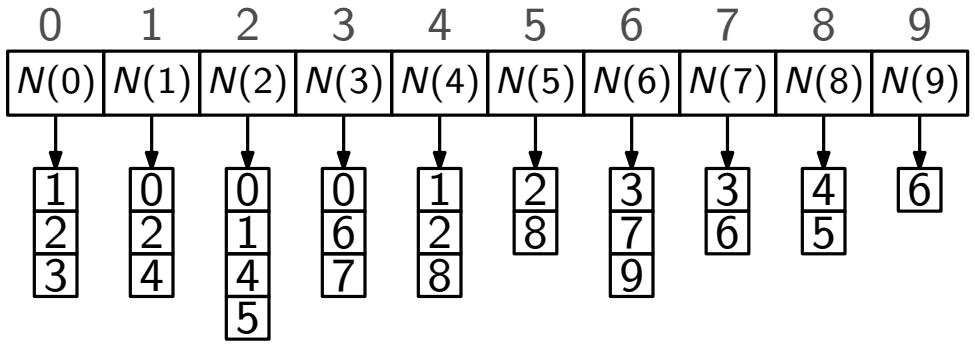


$\deg_{\text{in}} : [\mathbb{N}] = [0, 0, 2, 1, \ldots]$

sources : Queue

Wiederholung: Repräsentation von Graphen

Adjazenzliste



- 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

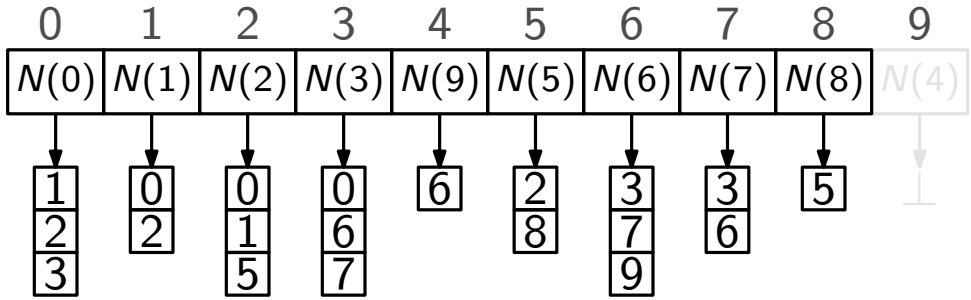
Naiver Ansatz: lösche

- 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

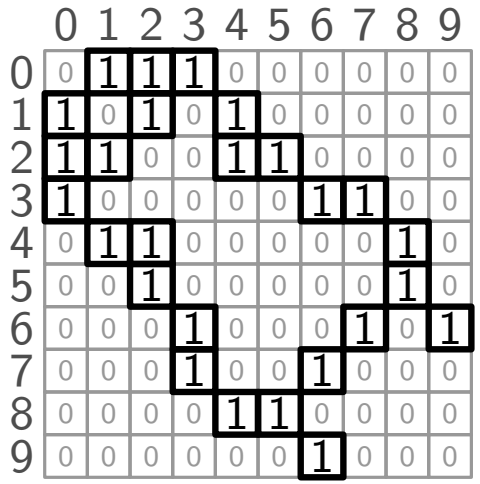
Gesamtlaufzeit: $\Theta(\deg(v))$

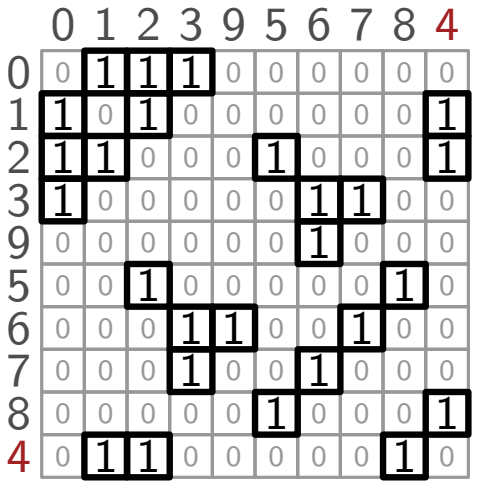
Und auf gerichteten Graphen?

Knoten löschen

Adjazenzmatrix

- swap mit hinterstem Element
  - in beiden Dimensionen





Laufzeit: $\Theta(n)$

Knoten löschen?

Problemstellung

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

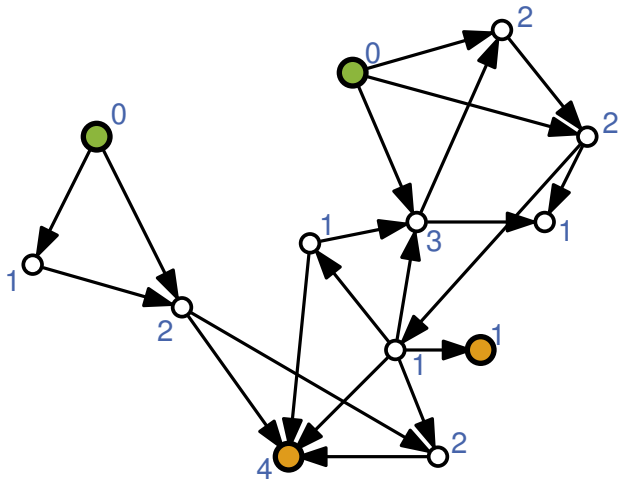
Algorithmus

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

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

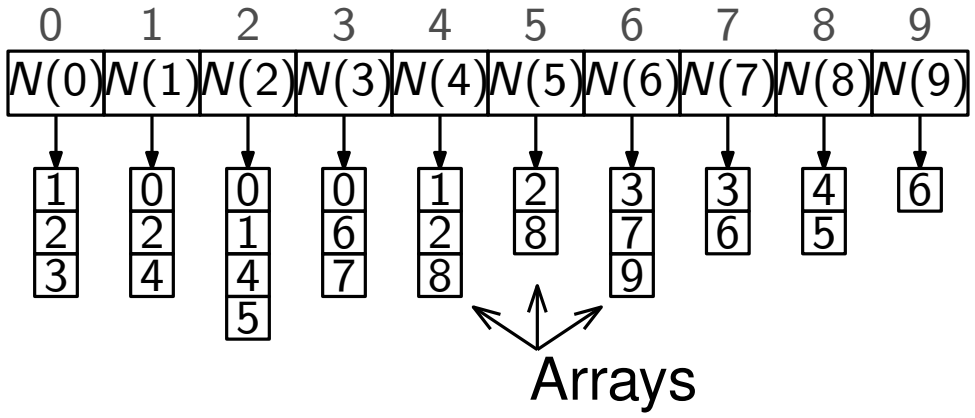
Adjazenzliste

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. $N(v) \cap N(w)$)