Algorithmen I

Tutorium 1 - 8. Sitzung

Dennis Felsing

dennis.felsing@student.kit.edu
www.stud.uni-karlsruhe.de/~ubcqr/algo

2011-06-06

Überblick

Graphen

Begriffe

Graphen

Begriffe

Graph $G = (V, E)$, **Knoten** V, **Kanten** $E \subseteq V \times V$

Graphen

Begriffe

Graph $G = (V, E)$, **Knoten** V, **Kanten** $E \subseteq V \times V$
Gerichteter Graph, **Ungerichteter Graph**

Graphen

Begriffe

Graph $G = (V, E)$, **Knoten** V, **Kanten** $E \subseteq V \times V$
Gerichteter Graph, **Ungerichteter Graph**
Grad $d(v)$: Anzahl Kanten an Knoten v

Graphen

Begriffe

Graph $G = (V, E)$, **Knoten** V, **Kanten** $E \subseteq V \times V$
Gerichteter Graph, **Ungerichteter Graph**
Grad $d(v)$: Anzahl Kanten an Knoten v
Ausgangsgrad $d^+(v)$: Anzahl ausgehender Kanten an $v \in V$
Eingangsgrad $d^-(v)$: Anzahl eingehender Kanten an $v \in V$

Graphen

Begriffe

Graph $G = (V, E)$, **Knoten** V, **Kanten** $E \subseteq V \times V$
Gerichteter Graph, **Ungerichteter Graph**
Grad $d(v)$: Anzahl Kanten an Knoten v
Ausgangsgrad $d^+(v)$: Anzahl ausgehender Kanten an $v \in V$
Eingangsgrad $d^-(v)$: Anzahl eingehender Kanten an $v \in V$
Pfad (Weg): Folge von Knoten $v_1 \ldots v_n$ mit $(v_k, v_{k+1}) \in E$

Graphen

Begriffe

Graph $G = (V, E)$, **Knoten** V, **Kanten** $E \subseteq V \times V$
Gerichteter Graph, **Ungerichteter Graph**
Grad $d(v)$: Anzahl Kanten an Knoten v
Ausgangsgrad $d^+(v)$: Anzahl ausgehender Kanten an $v \in V$
Eingangsgrad $d^-(v)$: Anzahl eingehender Kanten an $v \in V$
Pfad (Weg): Folge von Knoten $v_1 ... v_n$ mit $(v_k, v_{k+1}) \in E$
Zusammenhängender Graph: $\forall v, w \in V : \exists$ Weg von v nach w

Graphen

Begriffe

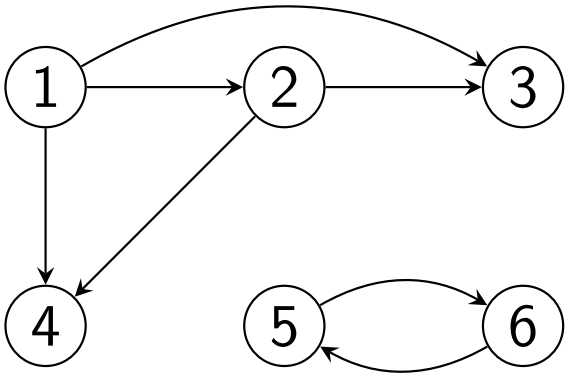
Graph $G = (V, E)$, **Knoten** V, **Kanten** $E \subseteq V \times V$
Gerichteter Graph, **Ungerichteter Graph**
Grad $d(v)$: Anzahl Kanten an Knoten v
Ausgangsgrad $d^+(v)$: Anzahl ausgehender Kanten an $v \in V$
Eingangsgrad $d^-(v)$: Anzahl eingehender Kanten an $v \in V$
Pfad (Weg): Folge von Knoten $v_1 ... v_n$ mit $(v_k, v_{k+1}) \in E$
Zusammenhängender Graph: $\forall v, w \in V : \exists$ Weg von v nach w
Zyklus: Weg mit gleichem Anfangs- und Endknoten

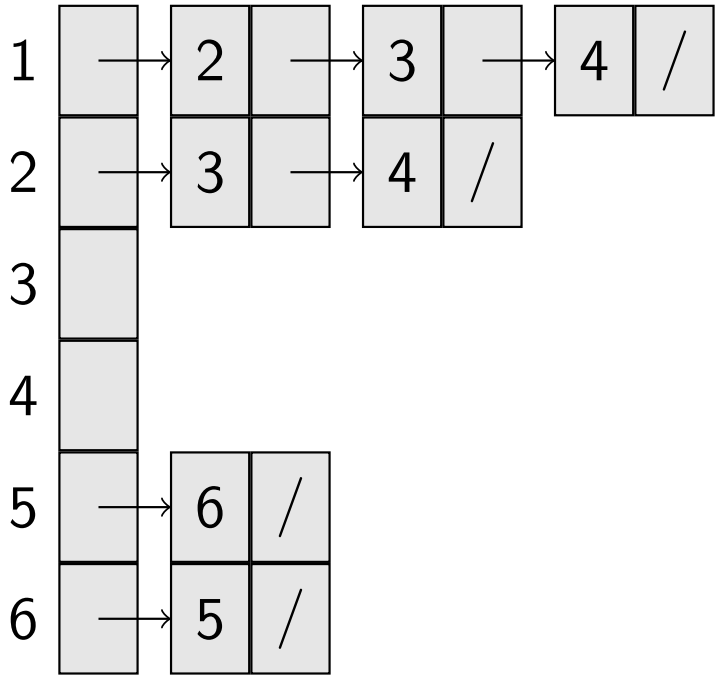
Graphen

Begriffe

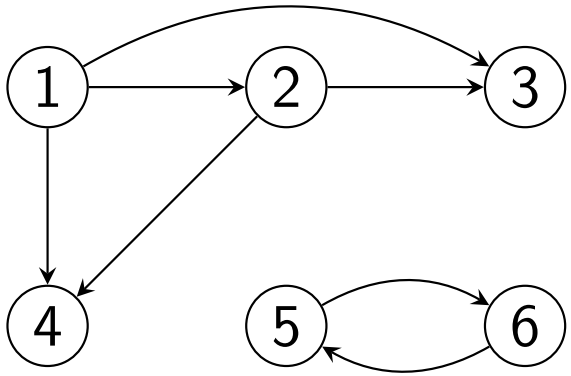
Graph $G = (V, E)$, **Knoten** V, **Kanten** $E \subseteq V \times V$
Gerichteter Graph, **Ungerichteter Graph**
Grad $d(v)$: Anzahl Kanten an Knoten v
Ausgangsgrad $d^+(v)$: Anzahl ausgehender Kanten an $v \in V$
Eingangsgrad $d^-(v)$: Anzahl eingehender Kanten an $v \in V$
Pfad (Weg): Folge von Knoten $v_1 ... v_n$ mit $(v_k, v_{k+1}) \in E$
Zusammenhängender Graph: $\forall v, w \in V : \exists$ Weg von v nach w
Zyklus: Weg mit gleichem Anfangs- und Endknoten
Eulerkreis:Zyklus, der alle Kanten genau ein mal benutzt
...

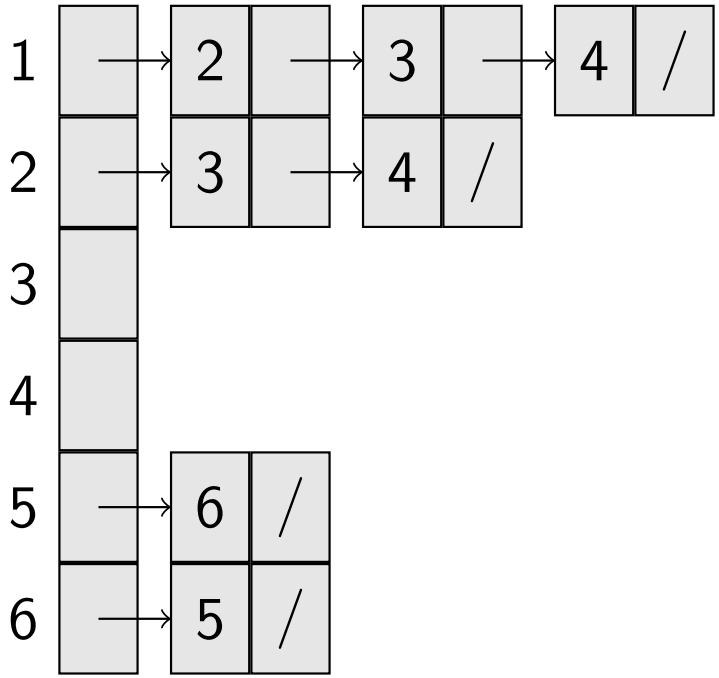
Adjazenzliste





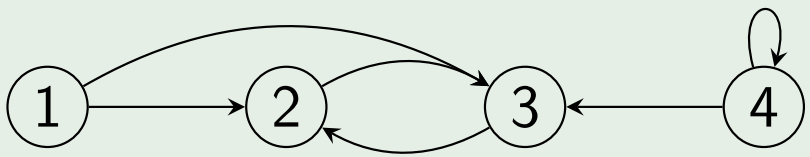
Adjazenzliste





Aufgabe

Gib die Adjazenzliste zu folgendem Graphen an:



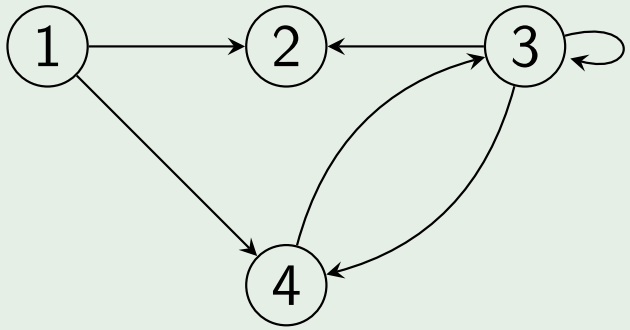
Adjazenzmatrix

Definition

Die **Adjazenzmatrix** eines Graphen $G = (V, E)$ ist eine Matrix mit

$$A_{ij} = \begin{cases} 1 & \text{falls } (i,j) \in E \\ 0 & \text{falls } (i,j) \notin E \end{cases}$$

Beispiel



$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

Adjazenzmatrix

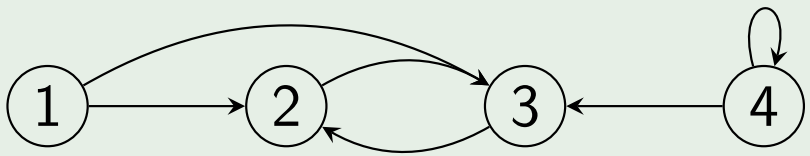
Definition

Die **Adjazenzmatrix** eines Graphen $G = (V, E)$ ist eine Matrix mit

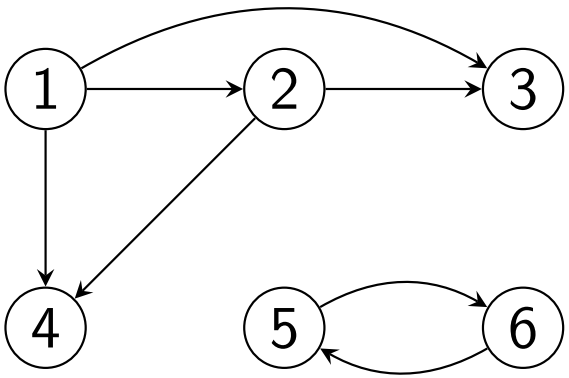
$$A_{ij} = \begin{cases} 1 & \text{falls } (i,j) \in E \\ 0 & \text{falls } (i,j) \notin E \end{cases}$$

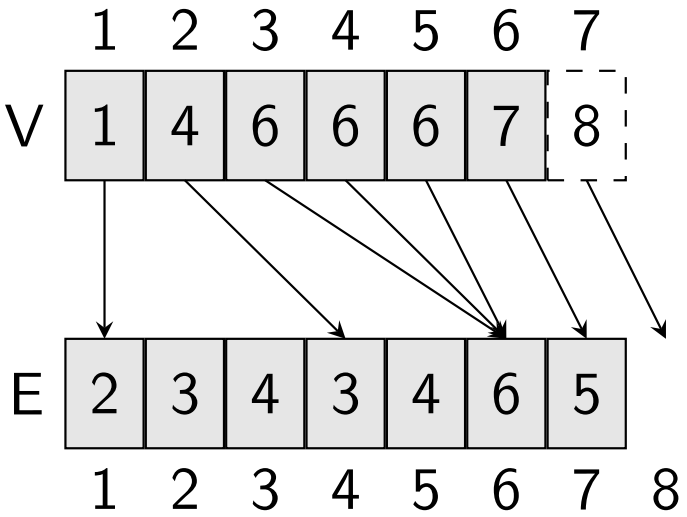
Aufgabe

Gib die Adjazenzmatrix zu folgendem Graphen an:

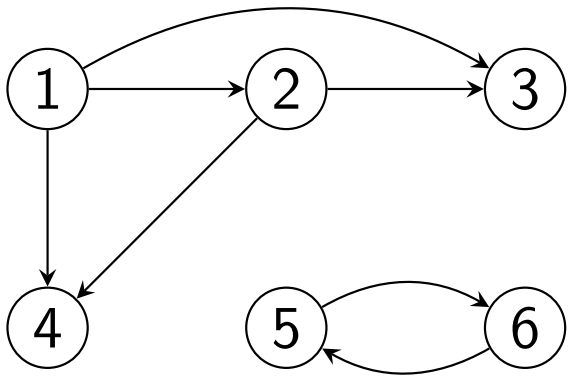


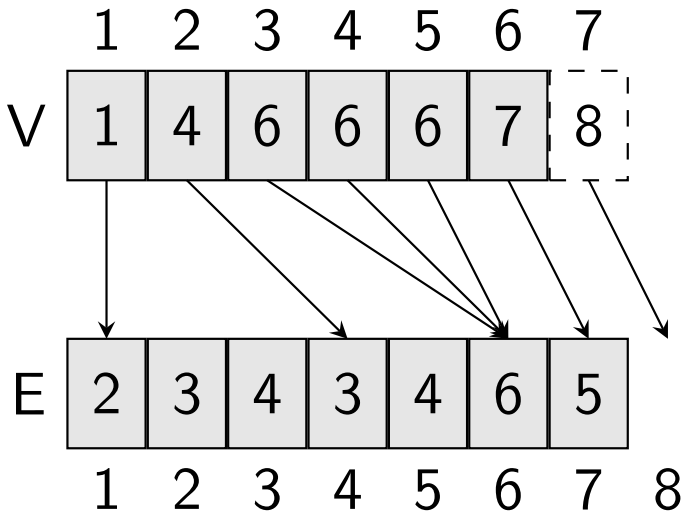
Adjazenzfeld





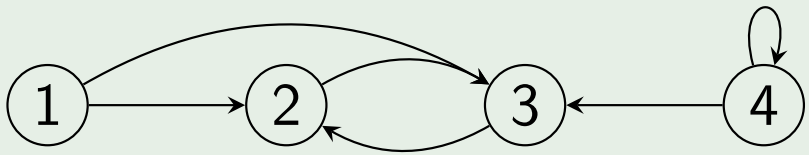
Adjazenzfeld





Aufgabe

Gib das Adjazenzfeld zu folgendem Graphen an:



Quiz

Sei $G = (V, E)$ ein gerichteter Graph mit $n := |V|$ und $m := |E|$ und A die zugehörige Adjazenzmatrix.

- Der Speicherverbrauch von Adjazenzliste liegt in

Quiz

Sei $G = (V, E)$ ein gerichteter Graph mit $n := |V|$ und $m := |E|$ und A die zugehörige Adjazenzmatrix.

- Der Speicherverbrauch von Adjazenzliste liegt in $\Theta(n + m)$
- Der Speicherverbrauch von Adjazenzmatrix liegt in

Quiz

Sei $G = (V, E)$ ein gerichteter Graph mit $n := |V|$ und $m := |E|$ und A die zugehörige Adjazenzmatrix.

- Der Speicherverbrauch von Adjazenzliste liegt in $\Theta(n + m)$
- Der Speicherverbrauch von Adjazenzmatrix liegt in $\Theta(n^2)$
- Der Speicherverbrauch von Adjazenzfeld liegt in

Quiz

Sei $G = (V, E)$ ein gerichteter Graph mit $n := |V|$ und $m := |E|$ und A die zugehörige Adjazenzmatrix.

- Der Speicherverbrauch von Adjazenzliste liegt in $\Theta(n + m)$
- Der Speicherverbrauch von Adjazenzmatrix liegt in $\Theta(n^2)$
- Der Speicherverbrauch von Adjazenzfeld liegt in $\Theta(n + m)$
- Einfügen von Kanten geht in Adjazenzlisten in $\Theta(1)$

Quiz

Sei $G = (V, E)$ ein gerichteter Graph mit $n := |V|$ und $m := |E|$ und A die zugehörige Adjazenzmatrix.

- Der Speicherverbrauch von Adjazenzliste liegt in $\Theta(n + m)$
- Der Speicherverbrauch von Adjazenzmatrix liegt in $\Theta(n^2)$
- Der Speicherverbrauch von Adjazenzfeld liegt in $\Theta(n + m)$
- Einfügen von Kanten geht in Adjazenzlisten in $\Theta(1)$ ✓
- Suchen nach Kanten geht in Adjazenzlisten in $\Theta(1)$

Quiz

Sei $G = (V, E)$ ein gerichteter Graph mit $n := |V|$ und $m := |E|$ und A die zugehörige Adjazenzmatrix.

- Der Speicherverbrauch von Adjazenzliste liegt in $\Theta(n + m)$
- Der Speicherverbrauch von Adjazenzmatrix liegt in $\Theta(n^2)$
- Der Speicherverbrauch von Adjazenzfeld liegt in $\Theta(n + m)$
- Einfügen von Kanten geht in Adjazenzlisten in $\Theta(1)$ ✓
- Suchen nach Kanten geht in Adjazenzlisten in $\Theta(1)$ ×
- Was bedeutet der Wert $n = (A^k)_{ij}$?

Quiz

Sei $G = (V, E)$ ein gerichteter Graph mit $n := |V|$ und $m := |E|$ und A die zugehörige Adjazenzmatrix.

- Der Speicherverbrauch von Adjazenzliste liegt in $\Theta(n + m)$
- Der Speicherverbrauch von Adjazenzmatrix liegt in $\Theta(n^2)$
- Der Speicherverbrauch von Adjazenzfeld liegt in $\Theta(n + m)$
- Einfügen von Kanten geht in Adjazenzlisten in $\Theta(1)$ ✓
- Suchen nach Kanten geht in Adjazenzlisten in $\Theta(1)$ ×
- Was bedeutet der Wert $n = (A^k)_{ij}$? Es gibt n Wege der Länge k von i nach j.
- A ist symmetrisch $\Leftrightarrow$

Quiz

Sei $G = (V, E)$ ein gerichteter Graph mit $n := |V|$ und $m := |E|$ und A die zugehörige Adjazenzmatrix.

- Der Speicherverbrauch von Adjazenzliste liegt in $\Theta(n + m)$
- Der Speicherverbrauch von Adjazenzmatrix liegt in $\Theta(n^2)$
- Der Speicherverbrauch von Adjazenzfeld liegt in $\Theta(n + m)$
- Einfügen von Kanten geht in Adjazenzlisten in $\Theta(1)$ ✓
- Suchen nach Kanten geht in Adjazenzlisten in $\Theta(1)$ ×
- Was bedeutet der Wert $n = (A^k)_{ij}$? Es gibt n Wege der Länge k von i nach j.
- A ist symmetrisch $\Leftrightarrow$ G ist ungerichtet
- Einfügen von Kanten geht in Adjazenzfeldern in $\Theta(1)$

Quiz

Sei $G = (V, E)$ ein gerichteter Graph mit $n := |V|$ und $m := |E|$ und A die zugehörige Adjazenzmatrix.

- Der Speicherverbrauch von Adjazenzliste liegt in $\Theta(n + m)$
- Der Speicherverbrauch von Adjazenzmatrix liegt in $\Theta(n^2)$
- Der Speicherverbrauch von Adjazenzfeld liegt in $\Theta(n + m)$
- Einfügen von Kanten geht in Adjazenzlisten in $\Theta(1)$ ✓
- Suchen nach Kanten geht in Adjazenzlisten in $\Theta(1)$ ×
- Was bedeutet der Wert $n = (A^k)_{ij}$? Es gibt n Wege der Länge k von i nach j.
- A ist symmetrisch $\Leftrightarrow$ G ist ungerichtet
- Einfügen von Kanten geht in Adjazenzfeldern in $\Theta(1)$ ×

Quiz

DAG (Directed Acyclic Graph) bezeichnet einen gerichteten Graphen ohne Zykel

- DAGs lassen sich in einer Adjazenzmatrix als obere Dreiecksmatrix repräsentieren

Quiz

DAG (Directed Acyclic Graph) bezeichnet einen gerichteten Graphen ohne Zykel

- DAGs lassen sich in einer Adjazenzmatrix als obere Dreiecksmatrix repräsentieren ✓
- Der Graph zur Adjazenzmatrix

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

  ist zusammenhängend

Quiz

DAG (Directed Acyclic Graph) bezeichnet einen gerichteten Graphen ohne Zykel

- DAGs lassen sich in einer Adjazenzmatrix als obere Dreiecksmatrix repräsentieren ✓
- Der Graph zur Adjazenzmatrix

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

  ist zusammenhängend ×
- Für jeden Knoten n eines gerichteten Graphen ist Ausgangsgrad $d^+(n) = d^-(n)$

Quiz

DAG (Directed Acyclic Graph) bezeichnet einen gerichteten Graphen ohne Zykel

- DAGs lassen sich in einer Adjazenzmatrix als obere Dreiecksmatrix repräsentieren ✓
- Der Graph zur Adjazenzmatrix

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

  ist zusammenhängend ×
- Für jeden Knoten n eines gerichteten Graphen ist Ausgangsgrad $d^+(n) = d^-(n)$ ×
- Für keinen Knoten n eines gerichteten Graphen ist Ausgangsgrad $d^+(n) = d^-(n)$

Quiz

DAG (Directed Acyclic Graph) bezeichnet einen gerichteten Graphen ohne Zykel

- DAGs lassen sich in einer Adjazenzmatrix als obere Dreiecksmatrix repräsentieren ✓
- Der Graph zur Adjazenzmatrix

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

  ist zusammenhängend ×
- Für jeden Knoten n eines gerichteten Graphen ist Ausgangsgrad $d^+(n) = d^-(n)$ ×
- Für keinen Knoten n eines gerichteten Graphen ist Ausgangsgrad $d^+(n) = d^-(n)$ ×

Aufgabe

Entwerfe einen Algorithmus, der eine Liste von Kanten in eine entsprechende Darstellung als Adjazenzfeld umwandelt.

Kreativaufgabe - Dynamisiertes Adjazenzfeld

Entwickle eine Graphenrepräsentation mit folgenden Eigenschaften:

- Stabile und eindeutige Knoten-IDs
- Eindeutige Kanten-IDs
- **Wahlfreier Zugriff** auf Knoten und Kanten in konstanter Zeit
- **Navigation** auf Knoten und Kanten in konstanter Zeit
- Amortisiert konstantes **Einfügen** von Knoten und Kanten
- Amortisiert konstantes **Entfernen** von Knoten und Kanten

Tiefensuche

```
DFS(G)
1   for each u ∈ G.V
2       u.color = WHITE
3       u.π = NIL
4   time = 0
5   for each u ∈ G.V
6       if u.color == WHITE
7           DFS-Visit(G, u)
```

Tiefensuche

DFS-VISIT(G, u)

```
 1  time = time + 1 // Weißer Knoten u gerade entdeckt
 2  u.d = time
 3  u.color = GRAY
 4  for each v ∈ G.Adj[u]
 5      if v.color == WHITE // Erkunde Kante (u, v)
 6          v.π = u
 7          DFS-VISIT(G, v)
 8  u.color = BLACK // u ist fertig
 9  time = time + 1
10  u.f = time
```

Tiefensuche

Kantenklassifikation

- **Baumkante** (u, v): v wurde über (u, v) entdeckt
- **Rückwärtskante** (u, v): v ist (Baumkanten-)Vorgänger von u
- **Vorwärtskante** (u, v): u ist (Baumkanten-)Vorgänger von v, aber (u, v) keine Baumkante
- **Querkante** (u, v): Rest

Topologisches Sortieren

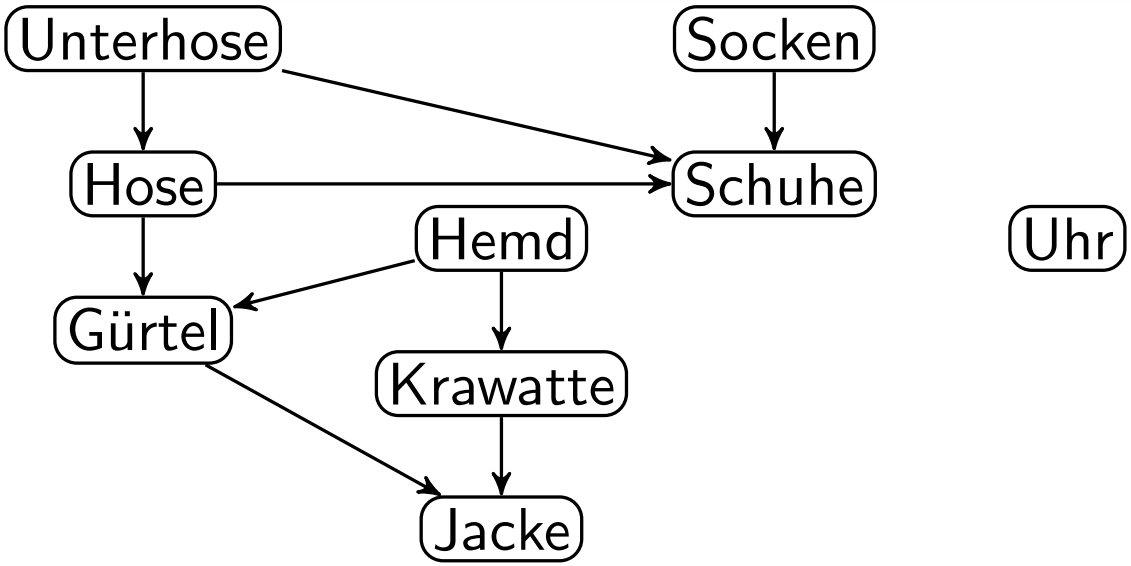
Definition

Eine **Topologische Sortierung** eines DAG $G = (V, E)$ ist eine lineare Anordnung seiner Knoten, so dass für alle Kanten (u, v) u vor v eingeordnet ist.

Topologisches Sortieren

Definition

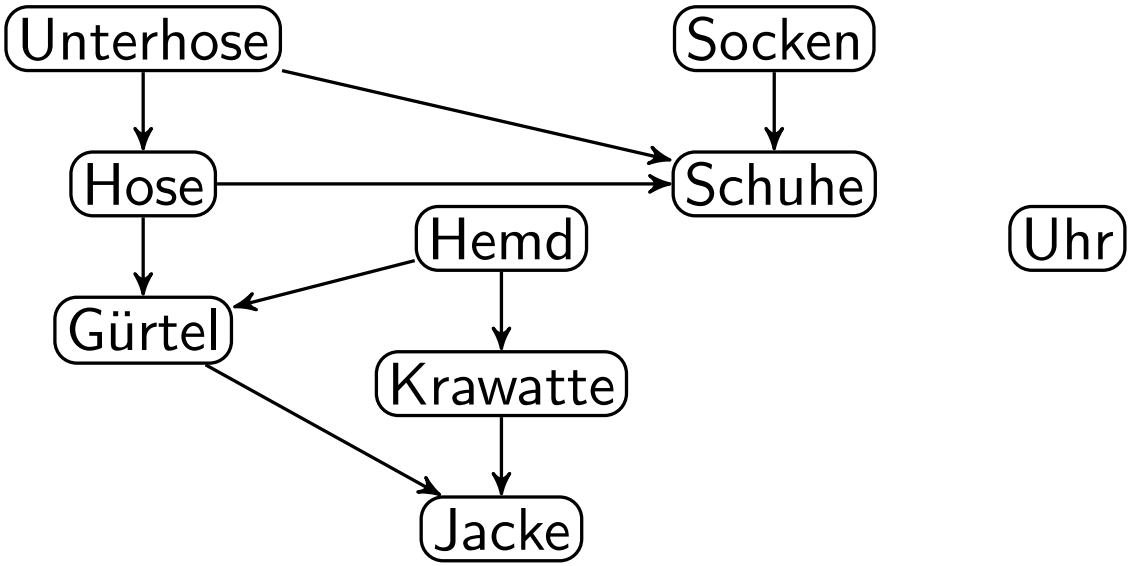
Eine **Topologische Sortierung** eines DAG $G = (V, E)$ ist eine lineare Anordnung seiner Knoten, so dass für alle Kanten (u, v) u vor v eingeordnet ist.

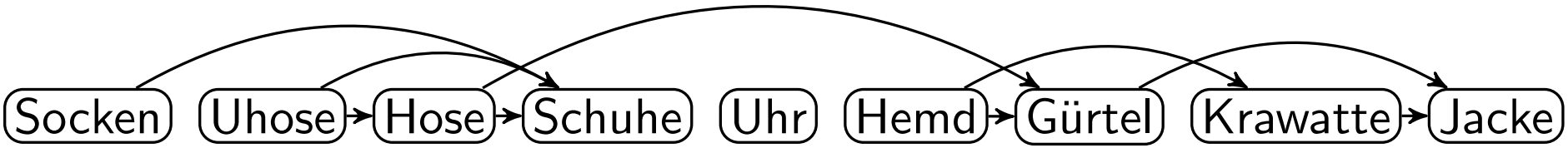


Topologisches Sortieren

Definition

Eine **Topologische Sortierung** eines DAG $G = (V, E)$ ist eine lineare Anordnung seiner Knoten, so dass für alle Kanten (u, v) u vor v eingeordnet ist.





Topologisches Sortieren

TOPOLOGICAL-SORT(u, v)

1 DFS(G) um $v.finished$ für alle $v \in V$ zu berechnen
2 Immer wenn *finished* gesetzt wird,
Knoten an Anfang einer Liste anfügen
3 **return** Entstandene Knotenliste

Topologisches Sortieren

TOPOLOGICAL-SORT(u, v)

1 DFS(G) um $v.finished$ für alle $v \in V$ zu berechnen
2 Immer wenn *finished* gesetzt wird,
Knoten an Anfang einer Liste anfügen
3 **return** Entstandene Knotenliste

Quiz

- Alle Graphen lassen sich topologisch sortieren.

Topologisches Sortieren

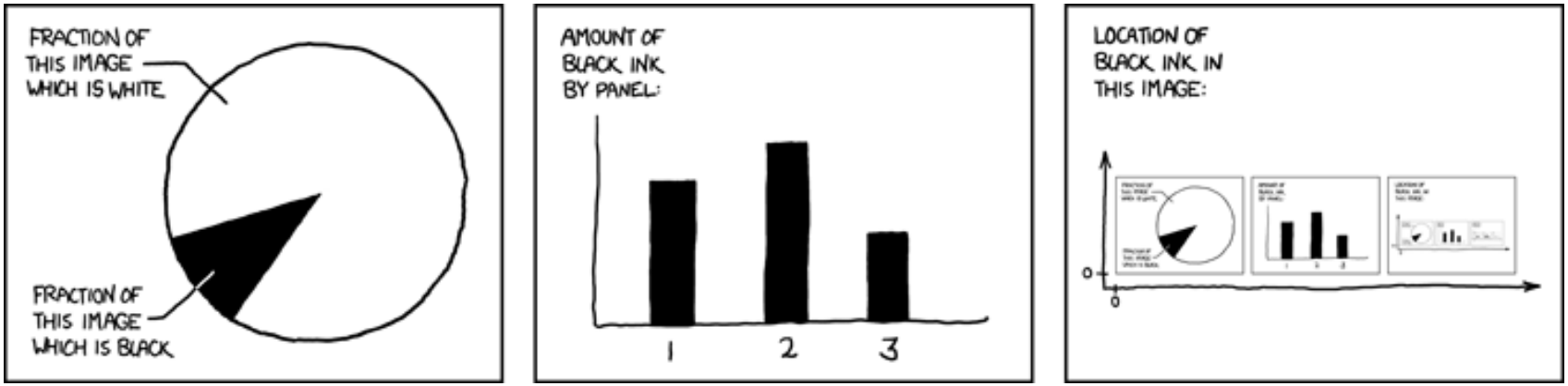
TOPOLOGICAL-SORT(u, v)

1 DFS(G) um $v.finished$ für alle $v \in V$ zu berechnen
2 Immer wenn *finished* gesetzt wird,
Knoten an Anfang einer Liste anfügen
3 **return** Entstandene Knotenliste

Quiz

- Alle Graphen lassen sich topologisch sortieren. ×

Übersicht


FRACTION OF THIS IMAGE WHICH IS WHITE
FRACTION OF THIS IMAGE WHICH IS BLACK
AMOUNT OF BLACK INK BY PANEL:
1
2
3
LOCATION OF BLACK INK IN THIS IMAGE:
0
0