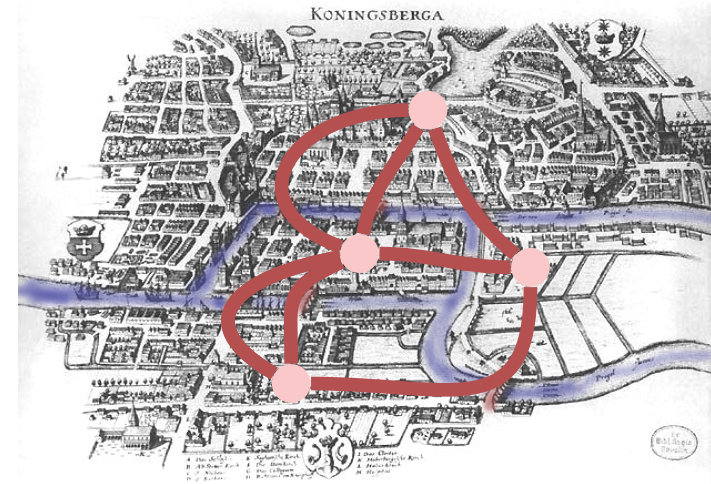


Königsberg 1736

12. Graphen

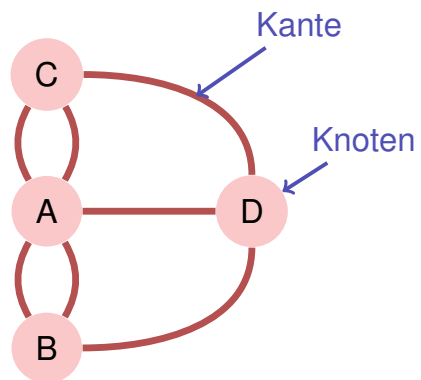
Notation, Repräsentation, Traversieren (DFS, BFS), Topologisches Sortieren, Ottman/Widmayer, Kap. 9.1 - 9.4, Cormen et al, Kap. 22



254

255

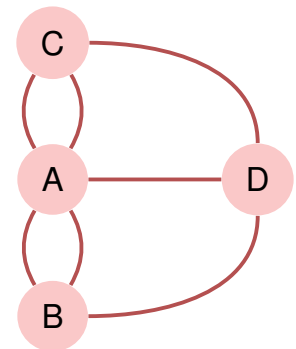
[Multi]Graph



Zyklen

- Gibt es einen Rundweg durch die Stadt (den Graphen), welcher jede Brücke (jede Kante) genau einmal benutzt?
- Euler (1736): nein.
- Solcher Rundweg (*Zyklus*) heisst *Eulerscher Kreis*.
- Eulerzyklus \Leftrightarrow jeder Knoten hat gerade Anzahl Kanten (jeder Knoten hat einen *geraden Grad*).

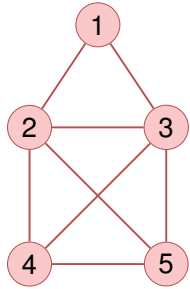
" \Rightarrow " ist klar, " \Leftarrow " ist etwas schwieriger



256

257

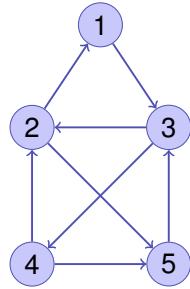
Notation



ungerichtet

$$V = \{1, 2, 3, 4, 5\}$$

$$E = \{\{1, 2\}, \{1, 3\}, \{2, 3\}, \{2, 4\}, \{2, 5\}, \{3, 4\}, \{3, 5\}, \{4, 5\}\}$$



gerichtet

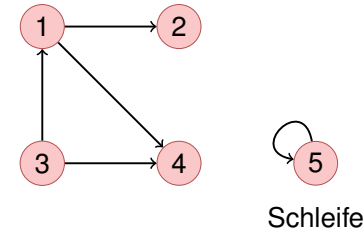
$$V = \{1, 2, 3, 4, 5\}$$

$$E = \{(1, 3), (2, 1), (2, 5), (3, 2), (3, 4), (4, 2), (4, 5), (5, 3)\}$$

258

Notation

Ein *gerichteter Graph* besteht aus einer Menge $V = \{v_1, \dots, v_n\}$ von Knoten (*Vertices*) und einer Menge $E \subseteq V \times V$ von Kanten (*Edges*). Gleiche Kanten dürfen nicht mehrfach enthalten sein.

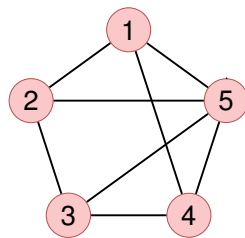


Schleife

259

Notation

Ein *ungerichteter Graph* besteht aus einer Menge $V = \{v_1, \dots, v_n\}$ von Knoten und einer Menge $E \subseteq \{\{u, v\} | u, v \in V\}$ von Kanten. Kanten dürfen nicht mehrfach enthalten sein.⁸



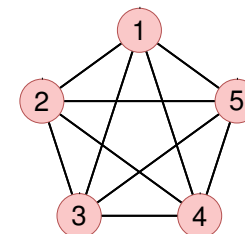
ungerichteter Graph

⁸Im Gegensatz zum Eingangsbeispiel – dann Multigraph genannt.

260

Notation

Ein ungerichteter Graph $G = (V, E)$ ohne Schleifen in dem jeder Knoten mit jedem anderen Knoten durch eine Kante verbunden ist, heisst *vollständig*.

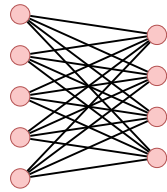


ein vollständiger ungerichteter Graph

261

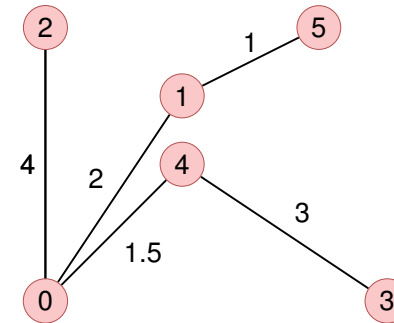
Notation

Ein Graph, bei dem V so in disjunkte U und W aufgeteilt werden kann, dass alle $e \in E$ einen Knoten in U und einen in W haben heisst *bipartit*.



Notation

Ein *gewichteter Graph* $G = (V, E, c)$ ist ein Graph $G = (V, E)$ mit einer *Kantengewichtsfunktion* $c: E \rightarrow \mathbb{R}$. $c(e)$ heisst *Gewicht* der Kante e .



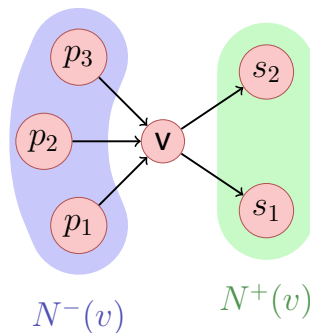
262

263

Notation

Für gerichtete Graphen $G = (V, E)$

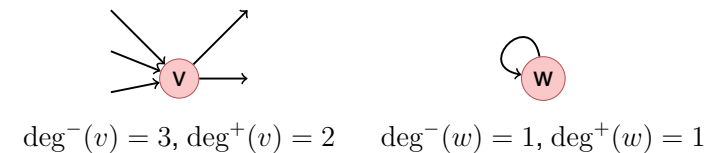
- $w \in V$ heisst *adjazent* zu $v \in V$, falls $(v, w) \in E$
- *Vorgängermenge* von $v \in V$: $N^-(v) := \{u \in V \mid (u, v) \in E\}$.
- *Nachfolgermenge*: $N^+(v) := \{u \in V \mid (v, u) \in E\}$



Notation

Für gerichtete Graphen $G = (V, E)$

- *Eingangsgrad*: $\deg^-(v) = |N^-(v)|$,
- *Ausgangsgrad*: $\deg^+(v) = |N^+(v)|$



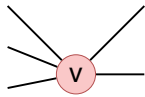
264

265

Notation

Für ungerichtete Graphen $G = (V, E)$:

- $w \in V$ heisst **adjazent** zu $v \in V$, falls $\{v, w\} \in E$
- **Nachbarschaft** von $v \in V$: $N(v) = \{w \in V \mid \{v, w\} \in E\}$
- **Grad** von v : $\deg(v) = |N(v)|$ mit Spezialfall Schleifen: erhöhen Grad um 2.



$$\deg(v) = 5$$



$$\deg(w) = 2$$

266

267

Beziehung zwischen Knotengraden und Kantenzahl

In jedem Graphen $G = (V, E)$ gilt

- 1 $\sum_{v \in V} \deg^-(v) = \sum_{v \in V} \deg^+(v) = |E|$, falls G gerichtet
- 2 $\sum_{v \in V} \deg(v) = 2|E|$, falls G ungerichtet.

Wege

- **Weg**: Sequenz von Knoten $\langle v_1, \dots, v_{k+1} \rangle$ so dass für jedes $i \in \{1 \dots k\}$ eine Kante von v_i nach v_{i+1} existiert.
- **Länge** des Weges: Anzahl enthaltene Kanten k .
- **Gewicht** des Weges (in gewichteten Graphen): $\sum_{i=1}^k c((v_i, v_{i+1}))$ (bzw. $\sum_{i=1}^k c(\{v_i, v_{i+1}\})$)
- **Pfad** (auch: einfacher Pfad): Weg der keinen Knoten mehrfach verwendet.

268

Zusammenhang

- Ungerichteter Graph heisst **zusammenhängend**, wenn für jedes Paar $v, w \in V$ ein verbindender Weg existiert.
- Gerichteter Graph heisst **stark zusammenhängend**, wenn für jedes Paar $v, w \in V$ ein verbindender Weg existiert.
- Gerichteter Graph heisst **schwach zusammenhängend**, wenn der entsprechende ungerichtete Graph zusammenhängend ist.

269

Einfache Beobachtungen

- Allgemein: $0 \leq |E| \in \mathcal{O}(|V|^2)$
- Zusammenhängender Graph: $|E| \in \Omega(|V|)$
- Vollständiger Graph: $|E| = \frac{|V| \cdot (|V|-1)}{2}$ (ungerichtet)
- Maximal $|E| = |V|^2$ (gerichtet), $|E| = \frac{|V| \cdot (|V|+1)}{2}$ (ungerichtet)

270

Zyklen

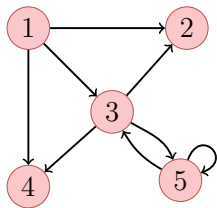
- **Zyklus:** Weg $\langle v_1, \dots, v_{k+1} \rangle$ mit $v_1 = v_{k+1}$
- **Kreis:** Zyklus mit paarweise verschiedenen v_1, \dots, v_k , welcher keine Kante mehrfach verwendet.
- **Kreisfrei (azyklisch):** Graph ohne jegliche Kreise.

Eine Folgerung: Ungerichtete Graphen können keinen Kreis der Länge 2 enthalten (Schleifen haben Länge 1).

271

Repräsentation mit Matrix

Graph $G = (V, E)$ mit Knotenmenge v_1, \dots, v_n gespeichert als **Adjazenzmatrix** $A_G = (a_{ij})_{1 \leq i, j \leq n}$ mit Einträgen aus $\{0, 1\}$. $a_{ij} = 1$ genau dann wenn Kante von v_i nach v_j .



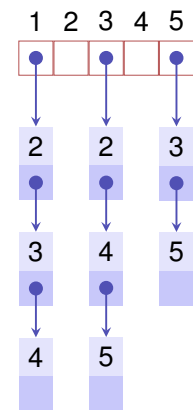
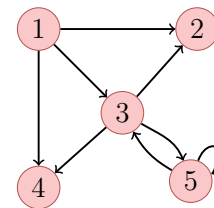
$$\begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix}$$

Speicherbedarf $\Theta(|V|^2)$. A_G ist symmetrisch, wenn G ungerichtet.

272

Repräsentation mit Liste

Viele Graphen $G = (V, E)$ mit Knotenmenge v_1, \dots, v_n haben deutlich weniger als n^2 Kanten. Repräsentation mit **Adjazenzliste**: Array $A[1], \dots, A[n]$, A_i enthält verkettete Liste aller Knoten in $N^+(v_i)$.



Speicherbedarf $\Theta(|V| + |E|)$.

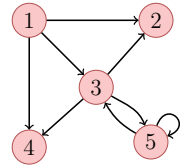
273

Laufzeiten einfacher Operationen

| Operation | Matrix | Liste |
|--|---------------|--------------------|
| Nachbarn/Nachfolger von $v \in V$ finden | $\Theta(n)$ | $\Theta(\deg^+ v)$ |
| $v \in V$ ohne Nachbar/Nachfolger finden | $\Theta(n^2)$ | $\Theta(n)$ |
| $(u, v) \in E$? | $\Theta(1)$ | $\Theta(\deg^+ v)$ |
| Kante einfügen | $\Theta(1)$ | $\Theta(1)$ |
| Kante löschen | $\Theta(1)$ | $\Theta(\deg^+ v)$ |

Adjazenzmatrizen multipliziert

$$B := A_G^2 = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix}^2 = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 2 \end{pmatrix}$$



274

275

Interpretation

Theorem

Sei $G = (V, E)$ ein Graph und $k \in \mathbb{N}$. Dann gibt das Element $a_{i,j}^{(k)}$ der Matrix $(a_{i,j}^{(k)})_{1 \leq i,j \leq n} = (A_G)^k$ die Anzahl der Wege mit Länge k von v_i nach v_j an.

Beweis

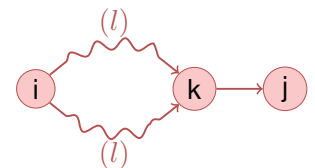
Per Induktion.

Anfang: Klar für $k = 1$. $a_{i,j} = a_{i,j}^{(1)}$.

Hypothese: Aussage wahr für alle $k \leq l$

Schritt ($l \rightarrow l + 1$):

$$a_{i,j}^{(l+1)} = \sum_{k=1}^n a_{i,k}^{(l)} \cdot a_{k,j}$$



$a_{k,j} = 1$ g.d.w. Kante von k nach j , 0 sonst. Summe zählt die Anzahl Wege der Länge l vom Knoten v_i zu allen Knoten v_k welche direkte Verbindung zu Knoten v_j haben, also alle Wege der Länge $l + 1$.

276

277

Beispiel: Kürzester Weg

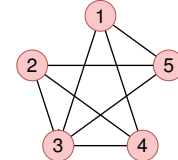
Frage: existiert Weg von i nach j ? Wie lang ist der kürzeste Weg?

Antwort: Potenziere A_G bis für ein $k < n$ gilt $a_{i,j}^{(k)} > 0$. k gibt die Weglänge des kürzesten Weges. Wenn $a_{i,j}^{(k)} = 0$ für alle $1 \leq k < n$, so gibt es keinen Weg von i nach j .

Beispiel: Anzahl Dreiecke

Frage: Wie viele Dreieckswege enthält ein ungerichteter Graph?

Antwort: Entferne alle Zyklen (Diagonaleinträge). Berechne A_G^3 . $a_{ii}^{(3)}$ bestimmt die Anzahl Wege der Länge 3, die i enthalten. Es gibt 6 verschiedene Permutationen eines Dreiecks. Damit Anzahl Dreiecke: $\sum_{i=1}^n a_{ii}^{(3)} / 6$.

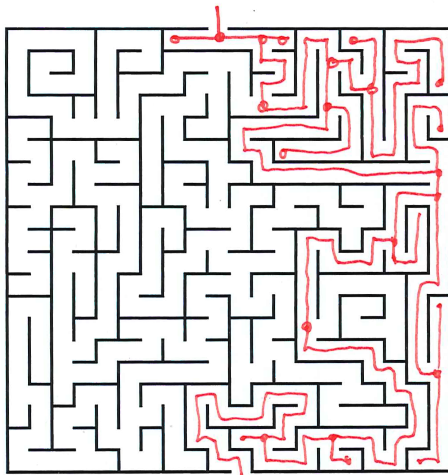


$$\begin{pmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{pmatrix}^3 = \begin{pmatrix} 4 & 4 & 8 & 8 & 8 \\ 4 & 4 & 8 & 8 & 8 \\ 8 & 8 & 8 & 8 & 8 \\ 8 & 8 & 8 & 4 & 4 \\ 8 & 8 & 8 & 4 & 4 \end{pmatrix} \Rightarrow 24/6 = 4 \text{ Dreiecke.}$$

278

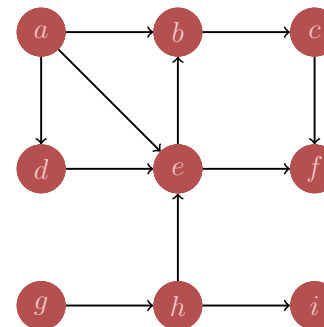
279

Tiefensuche



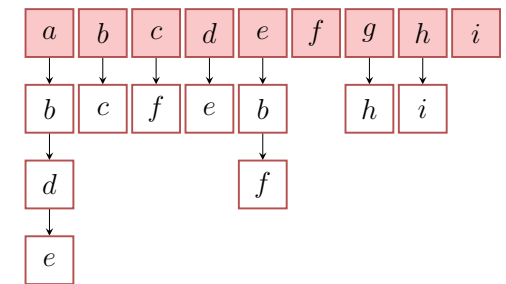
Graphen Traversieren: Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.



Reihenfolge
 $a, b, c, f, d, e, g, h, i$

Adjazenzliste



280

281

Algorithmus Tiefensuche DFS-Visit(G, v)

Input : Graph $G = (V, E)$, Knoten v .

Markiere v als besucht

```
foreach  $w \in N^+(v)$  do
  if  $\neg(w \text{ besucht})$  then
    DFS-Visit( $G, w$ )
```

Tiefensuche ab Knoten v . Laufzeit (ohne Rekursion): $\Theta(\deg^+ v)$

282

Algorithmus Tiefensuche DFS-Visit(G)

Input : Graph $G = (V, E)$

```
foreach  $v \in V$  do
  Markiere  $v$  als nicht besucht
```

```
foreach  $v \in V$  do
  if  $\neg(v \text{ besucht})$  then
    DFS-Visit( $G, v$ )
```

Tiefensuche für alle Knoten eines Graphen. Laufzeit
 $\Theta(|V| + \sum_{v \in V} (\deg^+(v) + 1)) = \Theta(|V| + |E|)$.

283

Iteratives DFS-Visit(G, v)

Input : Graph $G = (V, E)$

Stack $S \leftarrow \emptyset$; push(S, v)

while $S \neq \emptyset$ **do**

```
   $w \leftarrow \text{pop}(S)$ 
```

```
  if  $\neg(w \text{ besucht})$  then
```

```
    Markiere  $w$  besucht
```

```
    foreach  $(w, c) \in E$  do // (ggfs umgekehrt einfügen)
```

```
      if  $\neg(c \text{ besucht})$  then
```

```
        push( $S, c$ )
```

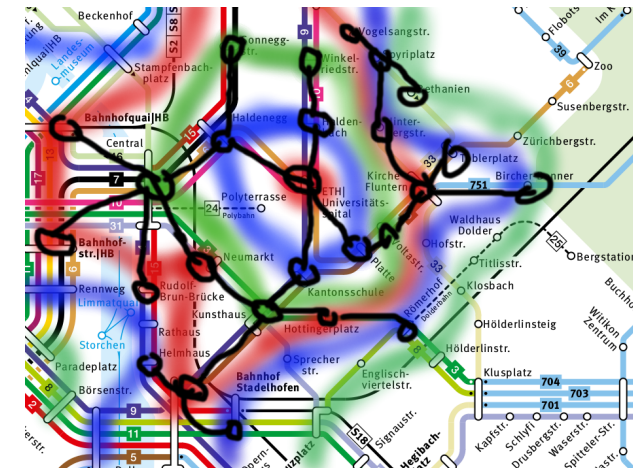
Stapelgrösse bis zu $|E|$, für jeden Knoten maximal Extraaufwand

$\Theta(\deg^+(w) + 1)$. Gesamt: $\Theta(|V| + |E|)$

Mit Aufruf aus obigem Rahmenprogramm: $\Theta(|V| + |E|)$

284

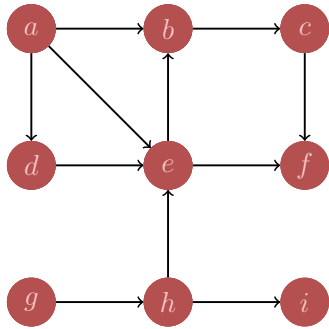
Breitensuche



285

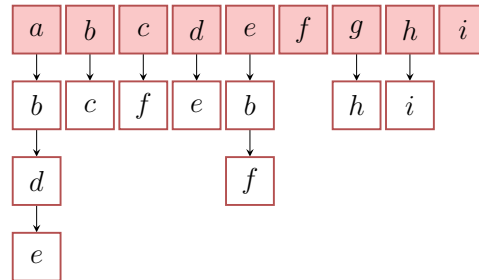
Graphen Traversieren: Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.



Reihenfolge
a, b, d, e, c, f, g, h, i

Adjazenzliste



Iteratives BFS-Visit(G, v)

Input : Graph $G = (V, E)$

```

Queue  $Q \leftarrow \emptyset$ 
Markiere  $v$  aktiv
enqueue( $Q, v$ )
while  $Q \neq \emptyset$  do
   $w \leftarrow$  dequeue( $Q$ )
  Markiere  $w$  besucht
  foreach  $c \in N^+(w)$  do
    if  $\neg(c$  besucht  $\vee c$  aktiv) then
      Markiere  $c$  aktiv
      enqueue( $Q, c$ )
    
```

- Algorithmus kommt mit $\mathcal{O}(|V|)$ Extraplatz aus.
- Gesamtlaufzeit mit Rahmenprogramm: $\Theta(|V| + |E|)$.

286

287

Topologisches Sortieren

Topologische Sortierung

| | A | B | C | D | E | F | G | H | I |
|----|--------|--------|--------|--------------|--------|-------|---|------|---|
| 1 | | Task 1 | Task 2 | Task 3 | Task 4 | Total | | Note | |
| 2 | TOTAL | 8 | 8 | 10 | 10 | 36 | | | |
| 3 | Arleen | 4 | 5 | 6 | 9 | 24 | | 4 | |
| 4 | Hans | 1 | 3 | 2 | 3 | 9 | | 1.5 | |
| 5 | Mike | 2 | 7 | 5 | 4 | 18 | | 3 | |
| 6 | Selina | 6 | 5 | 8 | 2 | 21 | | 3.5 | |
| 7 | | | | | | | | | |
| 8 | | | | Durchschnitt | | 18 | | 3 | |
| 9 | | | | | | | | | |
| 10 | | | | | | | | | |
| 11 | | | | | | | | | |
| 12 | | | | | | | | | |
| 13 | | | | | | | | | |
| 14 | | | | | | | | | |

Auswertungsreihenfolge?

Topologische Sortierung eines azyklischen gerichteten Graphen $G = (V, E)$:

Bijektive Abbildung

$$\text{ord} : V \rightarrow \{1, \dots, |V|\}$$

so dass

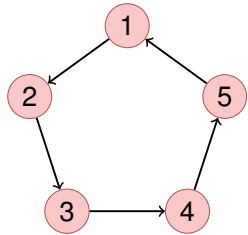
$$\text{ord}(v) < \text{ord}(w) \quad \forall (v, w) \in E.$$

Identifizieren Wert i mit dem Element $v_i := \text{ord}^{-1}(i)$. Topologische Sortierung $\hat{=} \langle v_1, \dots, v_{|V|} \rangle$.

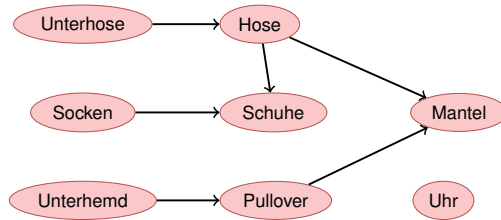
288

289

(Gegen-)Beispiele



Zyklischer Graph: kann nicht topologisch sortiert werden.



Eine mögliche Topologische Sortierung des Graphen:
Unterhemd, Pullover, Unterhose, Uhr, Hose, Mantel, Socken, Schuhe

Beobachtung

Theorem

Ein gerichteter Graph $G = (V, E)$ besitzt genau dann eine topologische Sortierung, wenn er kreisfrei ist

Beweis " \Rightarrow ": Wenn G einen Kreis besitzt, so besitzt er keine topologische Sortierung. Denn in einem Kreis $\langle v_{i_1}, \dots, v_{i_m} \rangle$ gälte $v_{i_1} < \dots < v_{i_m} < v_{i_1}$.

290

291

Induktiver Beweis Gegenrichtung

- Anfang ($n = 1$): Graph mit einem Knoten ohne Schleife ist topologisch sortierbar. Setze $\text{ord}(v_1) = 1$.
- Hypothese: Graph mit n Knoten kann topologisch sortiert werden.
- Schritt ($n \rightarrow n + 1$):
 - 1 G enthält einen Knoten v_q mit Eingangsgrad $\text{deg}^-(v_q) = 0$. Andernfalls verfolge iterativ Kanten rückwärts – nach spätestens $n + 1$ Iterationen würde man einen Knoten besuchen, welcher bereits besucht wurde. Widerspruch zur Zyklentreiheit.
 - 2 Graph ohne Knoten v_q und ohne dessen Eingangskanten kann nach Hypothese topologisch sortiert werden. Verwende diese Sortierung, setze $\text{ord}(v_i) \leftarrow \text{ord}(v_i) + 1$ für alle $i \neq q$ und setze $\text{ord}(v_q) \leftarrow 1$.

292

Algorithmus, vorläufiger Entwurf

Graph $G = (V, E)$. $d \leftarrow 1$

- 1 Traversiere von beliebigem Knoten rückwärts bis ein Knoten v_q mit Eingangsgrad 0 gefunden ist.
- 2 Wird kein Knoten mit Eingangsgrad 0 gefunden (n Schritte), dann Zyklus gefunden.
- 3 Setze $\text{ord}(v_q) \leftarrow d$.
- 4 Entferne v_q und seine Kanten von G .
- 5 Wenn $V \neq \emptyset$, dann $d \leftarrow d + 1$, gehe zu Schritt 1.

Laufzeit im schlechtesten Fall: $\Theta(|V|^2)$.

293

Verbesserung

Idee?

Berechne die Eingangsgrade der Knoten im Voraus und durchlaufe dann jeweils die Knoten mit Eingangsgrad 0 die Eingangsgrade der Nachfolgeknoten korrigierend.

Algorithmus Topological-Sort(G)

Input : Graph $G = (V, E)$.

Output : Topologische Sortierung ord

Stack $S \leftarrow \emptyset$

foreach $v \in V$ **do** $A[v] \leftarrow 0$

foreach $(v, w) \in E$ **do** $A[w] \leftarrow A[w] + 1$ // Eingangsgrade berechnen

foreach $v \in V$ with $A[v] = 0$ **do** push(S, v) // Merke Nodes mit Eingangsgrad 0

$i \leftarrow 1$

while $S \neq \emptyset$ **do**

$v \leftarrow \text{pop}(S)$; ord[v] $\leftarrow i$; $i \leftarrow i + 1$ // Wähle Knoten mit Eingangsgrad 0

foreach $(v, w) \in E$ **do** // Verringere Eingangsgrad der Nachfolger

$A[w] \leftarrow A[w] - 1$

if $A[w] = 0$ **then** push(S, w)

if $i = |V| + 1$ **then return** ord **else return** "Cycle Detected"

294

295

Algorithmus Korrektheit

Theorem

Sei $G = (V, E)$ ein gerichteter, kreisfreier Graph. Der Algorithmus TopologicalSort(G) berechnet in Zeit $\Theta(|V| + |E|)$ eine topologische Sortierung ord für G .

Beweis: folgt im wesentlichen aus vorigem Theorem:

- 1 Eingangsgrad verringern entspricht Knotenentfernen.
- 2 Im Algorithmus gilt für jeden Knoten v mit $A[v] = 0$ dass entweder der Knoten Eingangsgrad 0 hat oder dass zuvor alle Vorgänger einen Wert ord[u] $\leftarrow i$ zugewiesen bekamen und somit ord[v] $>$ ord[u] für alle Vorgänger u von v . Knoten werden nur einmal auf den Stack gelegt.
- 3 Laufzeit: Inspektion des Algorithmus (mit Argumenten wie beim Traversieren).

296

Algorithmus Korrektheit

Theorem

Sei $G = (V, E)$ ein gerichteter, nicht kreisfreier Graph. Der Algorithmus TopologicalSort(G) terminiert in Zeit $\Theta(|V| + |E|)$ und detektiert Zyklus.

Beweis: Sei $\langle v_{i_1}, \dots, v_{i_k} \rangle$ ein Kreis in G . In jedem Schritt des Algorithmus bleibt $A[v_{i_j}] \geq 1$ für alle $j = 1, \dots, k$. Also werden k Knoten nie auf den Stack gelegt und somit ist zum Schluss $i \leq |V| + 1 - k$.

Die Laufzeit des zweiten Teils des Algorithmus kann kürzer werden, jedoch kostet die Berechnung der Eingangsgrade bereits $\Theta(|V| + |E|)$.

297

Alternative: Algorithmus DFS-Topsort(G, v)

Input : Graph $G = (V, E)$, Knoten v , Knotenliste L .

if v aktiv **then**
 └ stop (Zyklus)

if v besucht **then**
 └ **return**

Markiere v aktiv

foreach $w \in N^+(v)$ **do**
 └ DFS-Topsort(G, w)

Markiere v besucht

Füge v am Anfang der Liste L ein

Rufe Algorithmus für jeden noch nicht besuchten Knoten auf.

Asymptotische Laufzeit $\Theta(|V| + |E|)$.