

# Algorithmen 1

## Tiefensuche: Brücken finden



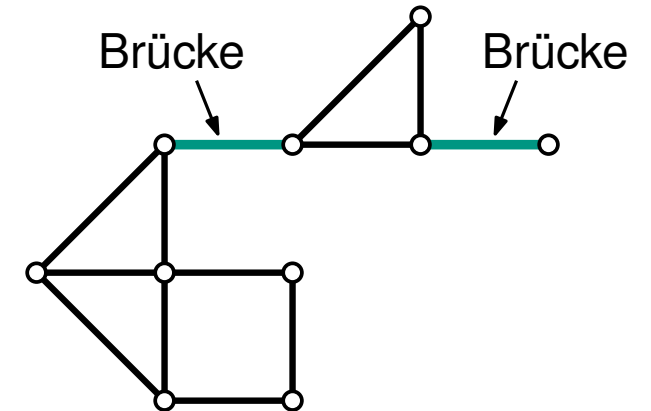
# Brücken in Graphen

## Definition

Sei  $G = (V, E)$  ein Graph. Eine Kante  $e = \{u, v\} \in E$  heißt **Brücke**, wenn  $u$  und  $v$  in  $G - e$  in verschiedenen Zusammenhangskomponenten liegen.

## Ziel heute

- finde alle Brücken
- lerne dabei die Tiefensuche (DFS) besser kennen



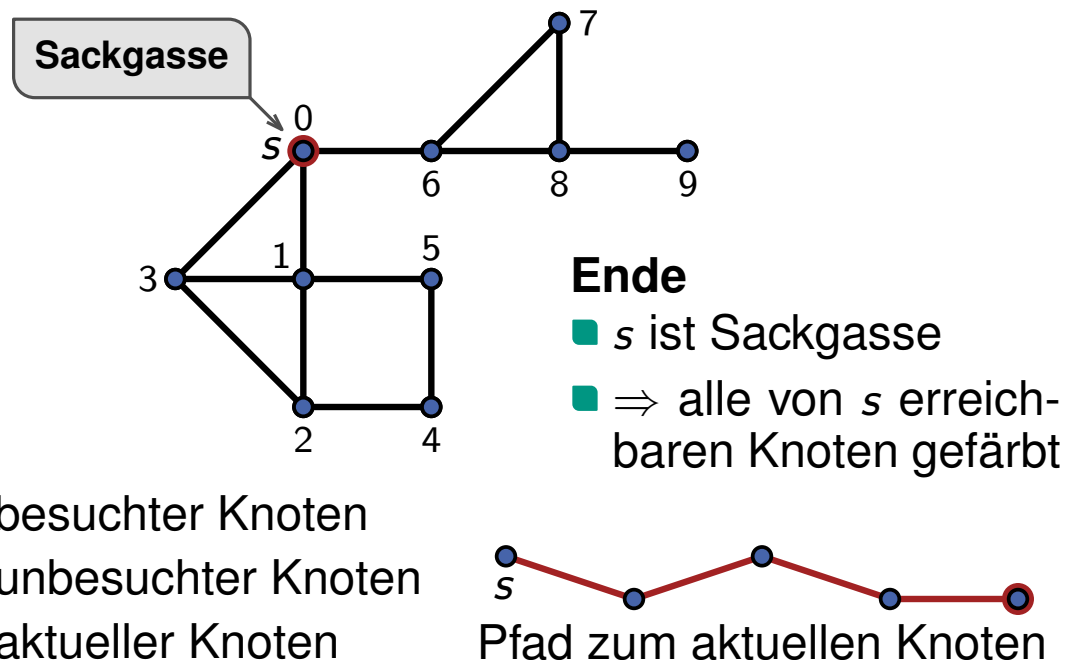
## Warum interessieren uns Brücken?

- interessante Graphtheoretische Struktur  $\rightarrow$  kleinstmögliche Schnitte
- zerlegt den Graphen in mehrere Komponenten  $\rightarrow$  gut für Teile und Herrsche

# Erinnerung: Graphtraversierung

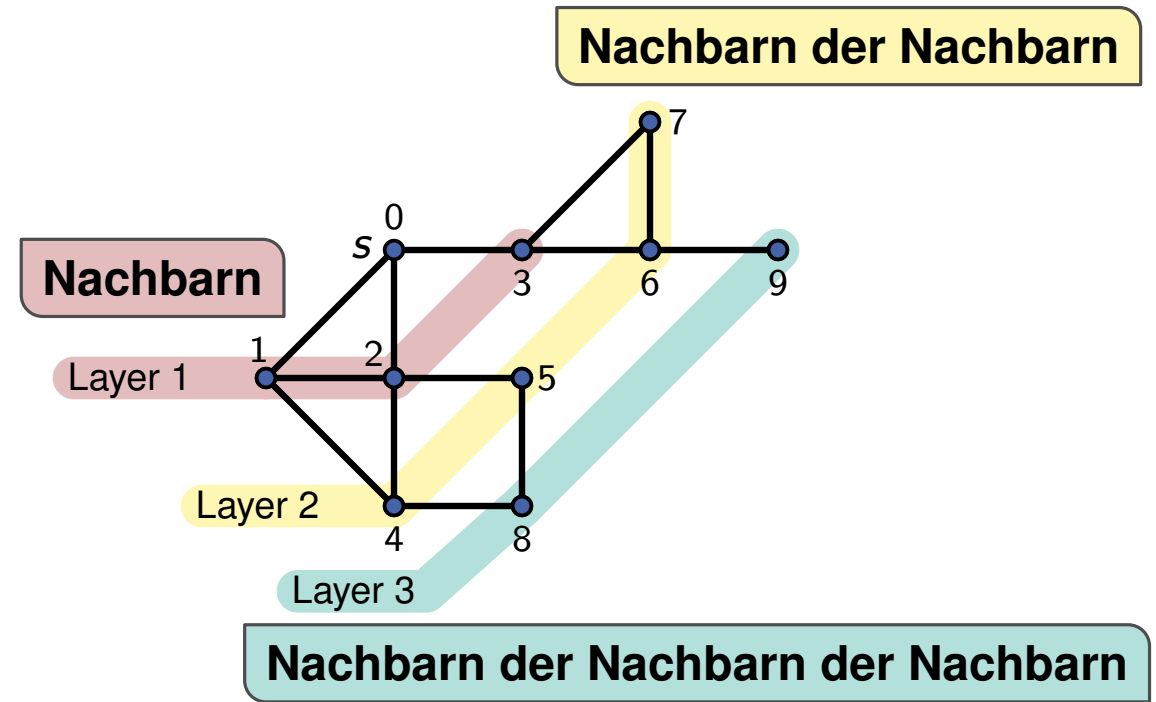
## Tiefensuche (DFS)

- starte bei einem Knoten  $s$
- laufe in jedem Schritt zu neuem Nachbarn
- Sackgasse (kein neuer Nachbar)  
→ Backtracking: zurück zum Vorgänger



## Breitensuche (BFS)

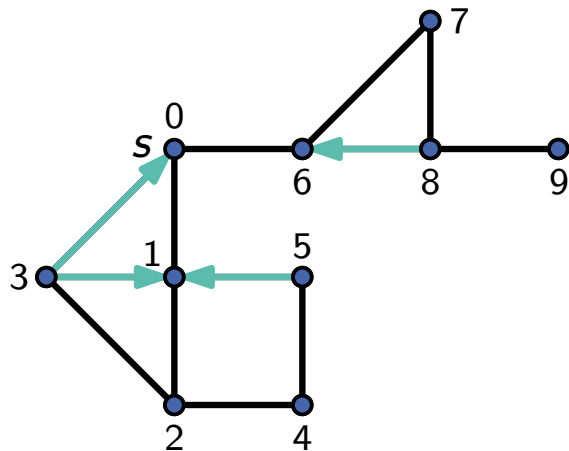
- starte bei einem Knoten  $s$
- besuche alle Nachbarn von  $s$
- dann alle Nachbarn der Nachbarn usw.



# DFS-Baum

## Beobachtung

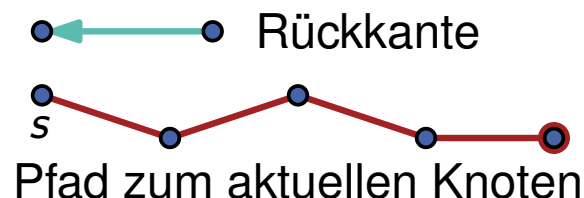
- beim Explorieren eines Knotens überspringen wir Kanten zu schon besuchten Knoten
- wir nennen sie auch **Rückkanten**
- nicht-übersprungene Kanten bilden einen Baum → **Baumkanten**



## Anmerkung: Rückkanten

- zeigen immer zu einem Vorgänger im DFS-Baum
- englisch: back edge
- gerichtete Graphen: weitere Kantentypen

- besuchter Knoten
- unbesuchter Knoten
- aktueller Knoten



# Charakterisierung von Brücken via DFS-Baum

## Brücken und Bäume

- in einem Baum ist jede Kante eine Brücke
- die Rückkanten sind entscheidend

### Anmerkung: Notation

- Knoten sind hier mit ihrer DFS-Nummer bezeichnet
- Rückkanten sehen wir als gerichtet an

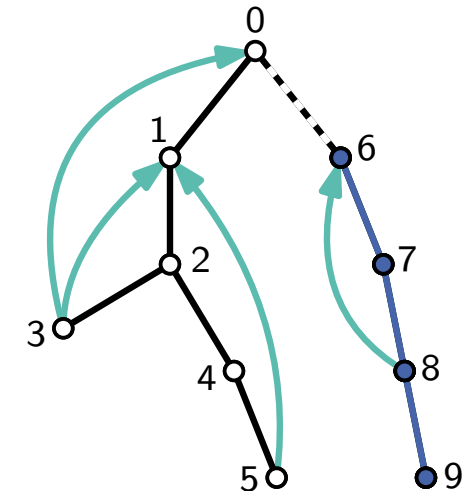
## Beispiel: Warum ist $\{0, 6\}$ eine Brücke?

- löschen von  $\{0, 6\}$  trennt im DFS-Baum den Teilbaum unter 6 vom Rest ab
- es gibt keine Rückkante aus dem Teilbaum zu einem Vorgänger von 6
- man erreicht insbesondere nicht mehr die Wurzel (0)

## Wann ist eine Baumkante $\{u, v\}$ eine Brücke?

- sei  $u$  der Elter und  $v$  das Kind
- $\{u, v\}$  ist Brücke  $\Leftrightarrow$  keine Rückkante aus dem Teilbaum unter  $v$  nach oberhalb von  $v$

**Außerdem:** Rückkanten sind nie Brücken



# Beweis der strukturellen Einsicht

## Theorem

(strukturell)

Eine Kante  $\{u, v\}$  ist genau dann eine Brücke, wenn

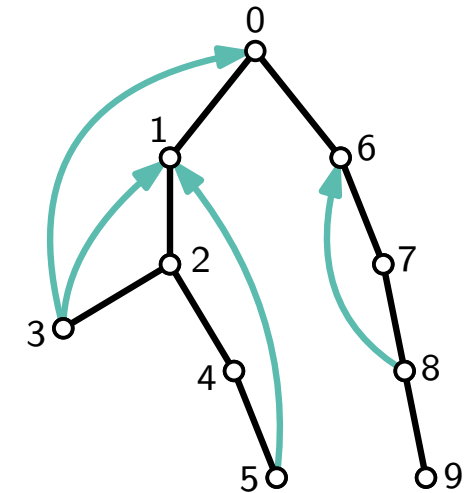
- $\{u, v\}$  ist eine Baumkante mit Elter  $u$  und Kind  $v$  und (1)
- keine Rückkante aus dem der Teilbaum unter  $v$  endet oberhalb von  $v$ . (2)

## Grober Plan: Fallunterscheidung

- Fall 1:  $\{u, v\}$  ist eine Rückkante  $\Rightarrow$  keine Brücke
- Fall 2:  $\{u, v\}$  ist eine Baumkante  $\Rightarrow$  [Brücke  $\Leftrightarrow$  (2) gilt]
  - Fall 2.1: (2) gilt nicht (es gibt Rückkante)  $\Rightarrow$  keine Brücke
  - Fall 2.2: (2) gilt (es gibt keine Rückkante)  $\Rightarrow$  Brücke

## Beachte

- die Fallunterscheidung deckt alle Fälle ab
- zeige: in jedem Fall ist die Aussage des Theorems erfüllt



# Beweis der strukturellen Einsicht

## Theorem

(strukturell)

Eine Kante  $\{u, v\}$  ist genau dann eine Brücke, wenn

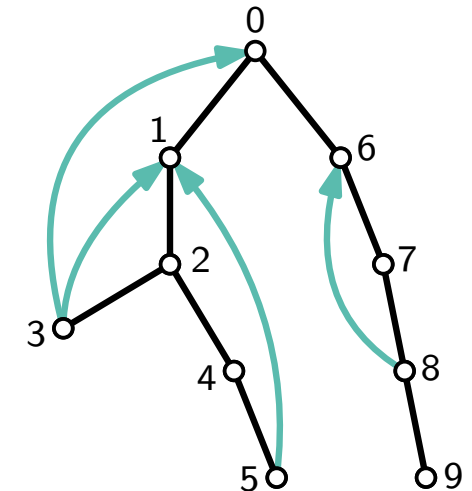
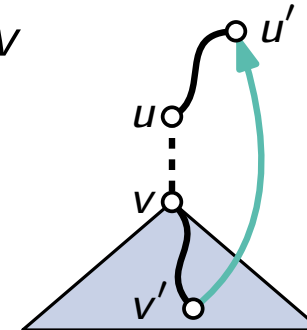
- $\{u, v\}$  ist eine Baumkante mit Elter  $u$  und Kind  $v$  und (1)
- keine Rückkante aus dem der Teilbaum unter  $v$  endet oberhalb von  $v$ . (2)

### Zeige für Fall 1: $\{u, v\}$ ist eine Rückkante $\Rightarrow$ keine Brücke

- es gibt einen Pfad zwischen  $u$  und  $v$  im DFS-Baum
- $u$  und  $v$  bleiben in der selben Komponente, wenn man  $\{u, v\}$  löscht

### Zeige für Fall 2.1: (2) gilt nicht (es gibt Rückkante) $\Rightarrow$ keine Brücke

- sei  $\{v', u'\}$  die Rückkante, sodass  $v'$  im Teilbaum von  $v$  und  $u'$  über  $v$  liegt
- (Baumpfad  $v, \dots, v'$ ) +  $\{v', u'\}$  + (Baumpfad  $u', \dots, u$ )  
 $\Rightarrow u$  und  $v$  bleiben in der selben Komponente



# Beweis der strukturellen Einsicht

## Theorem

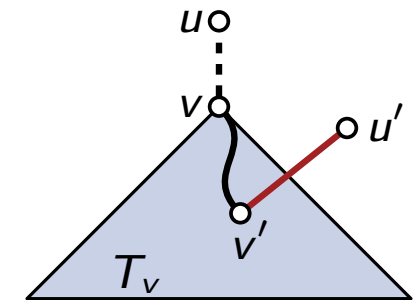
(strukturell)

Eine Kante  $\{u, v\}$  ist genau dann eine Brücke, wenn

- $\{u, v\}$  ist eine Baumkante mit Elter  $u$  und Kind  $v$  und (1)
- keine Rückkante aus dem der Teilbaum unter  $v$  endet oberhalb von  $v$ . (2)

**Zeige für Fall 2.2: (2) gilt (es gibt kein Rückkante)  $\Rightarrow$  Brücke**

- sei  $T_v = (V_v, E_v)$  der Teilbaum unter  $v$  (inklusive  $v$ )
- $\{u, v\}$  ist die einzige Kante zwischen  $T_v$  und Knoten außerhalb, denn:
  - angenommen es gibt weitere Kante  $\{u', v'\}$  mit  $v' \in V_v$  und  $u' \notin V_v$
  - $\{u', v'\}$  ist keine Baumkante, sonst wäre  $u'$  in  $T_v$
  - $\{u', v'\}$  ist also Rückkante und  $u'$  ist Vorgänger von  $v'$  im DFS-Baum
  - da  $u' \notin V_v$  muss  $u'$  auch oberhalb von  $v$  liegen  $\Rightarrow$  (2) gilt nicht ⚡
- $\{u, v\}$  löschen trennt  $T_v$  vom Rest des Graphen  $\Rightarrow \{u, v\}$  ist Brücke





# Von der strukturellen zur algorithmischen Sichtweise

## Theorem

(strukturell)

Eine Kante  $\{u, v\}$  ist genau dann eine Brücke, wenn

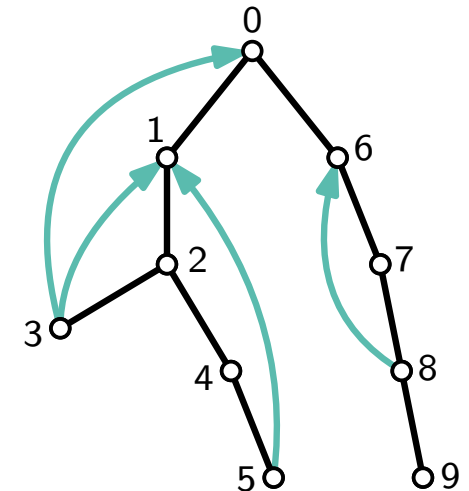
- $\{u, v\}$  ist eine Baumkante mit Elter  $u$  und Kind  $v$  und (1)
- keine Rückkante aus dem der Teilbaum unter  $v$  endet oberhalb von  $v$ . (2)

## Algorithmische Sichtweise

- (1) ist leicht zu überprüfen für eine Kante  $\{u, v\}$
- Problem bei (2): den ganzen Teilbaum unter  $v$  anschauen ist teuer  
 (linear in der Größe des Teilbaums  $\rightarrow$  potentiell quadratisch insgesamt)

## Plan im Folgenden

- definiere einen Wert  $low(v)$  für jeden Knoten  $v$
- sodass man an  $low(v)$  ablesen kann, ob (2) für  $v$  erfüllt ist
- berechne diese  $low(v)$  während der DFS gleich mit



# Algorithmische Umsetzung

## Definition

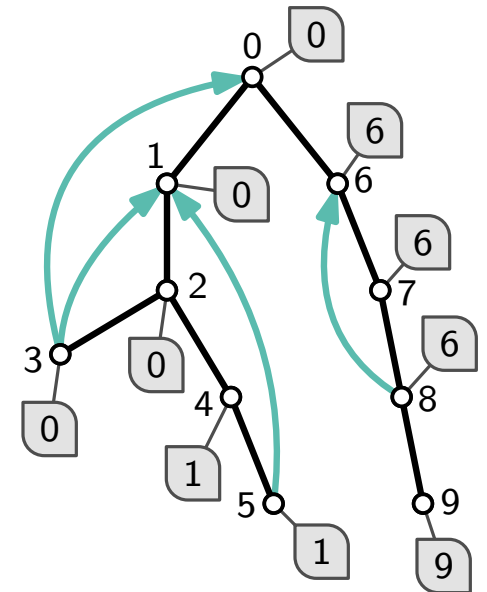
Sei  $\text{low}(v)$  das Minimum aus der DFS-Nummer von  $v$  und der kleinsten DFS-Nummer, die man von einem Knoten im Teilbaum unter  $v$  mit einer Rückkante erreichen kann.

## Wann ist eine Baumkante $\{u, v\}$ eine Brücke?

- keine Rückkante (2)  $\Rightarrow \text{low}(v) = \text{DFS-Nummer von } v$
- Rückkante (nicht 2)  $\Rightarrow \text{low}(v) < \text{DFS-Nummer von } v$
- also:  $\{u, v\}$  ist Brücke  $\Leftrightarrow \text{low}(v) = \text{DFS-Nummer von } v$

## Wie berechnen wir $\text{low}(v)$ ?

- können wir on-the-fly bei der DFS erledigen
- grob:  $\text{low}(v)$  ergibt sich aus
  - der DFS-Nummer von  $v$
  - den Rückkanten von  $v$
  - den Werten  $\text{low}(u_1), \dots, \text{low}(u_k)$  der Kinder  $u_1, \dots, u_k$  von  $v$



# Berechnung von $\text{low}(v)$

## Definition

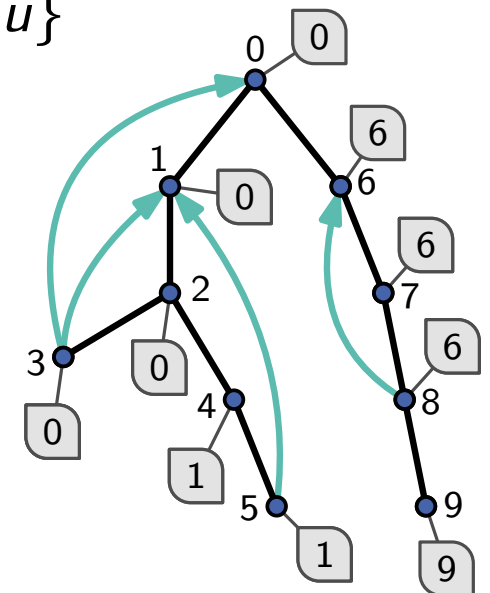
Sei  $\text{low}(v)$  das Minimum aus der DFS-Nummer von  $v$  und der kleinsten DFS-Nummer, die man von einem Knoten im Teilbaum unter  $v$  mit einer Rückkante erreichen kann.

## Situationen in denen wir $\text{low}(v)$ anpassen

- $v$  das erste Mal gesehen:  $\text{low}(v) := \text{DFS-Nummer von } v$
- Rückkante  $\{v, u\}$  gefunden:  $\text{low}(v) := \min\{\text{low}(v), \text{DFS-Nummer von } u\}$
- backtracken von einem Kind  $u$  von  $v$ :  $\text{low}(v) := \min\{\text{low}(v), \text{low}(u)\}$

## Korrektheit

- zeige: wenn man von  $v$  zum Elter backtracket, dann ist  $\text{low}(v)$  korrekt
- strukturelle Induktion:
  - es gilt für die Blätter (Anfang)
  - angenommen es gilt für die Kinder von  $v$  (Voraussetzung)
  - dann gilt es auch für  $v$  (Schritt)



# Algorithmus → Rekursive Implementierung

**DFS**(*Graph G, Node v, Node p*)

```

dfs[v] := next_dfs
next_dfs := next_dfs + 1
low[v] := dfs[v]
  
```

**Erinnerung: Situationen in denen wir  $low[v]$  anpassen**

- $v$  das erste Mal gesehen:  $low[v] := dfs[v]$
- Rückkante  $\{v, u\}$ :  $low[v] := \min\{low[v], dfs[u]\}$
- backtracken von Kind  $u$  zu  $v$ :  $low[v] := \min\{low[v], low[u]\}$

← erster Besuch bei  $v$

**for** *Node u* in  $N(v)$  **do**

**if**  $u = p$  **then continue**

← Achtung: wir müssen  $u$  überspringen, wenn  $u$  der Elter von  $v$  ist

**if**  $dfs[u] = \infty$  **then**

(sonst interpretieren wir die Baumkante von  $u$  nach  $v$  als Rückkante von  $v$  nach  $u$ )

**DFS**( $G, u, v$ )

$low[v] = \min\{low[v], low[u]\}$

← DFS für Kind  $u$  fertig → backtracken von  $u$  zu  $v$

**else**

$low[v] = \min\{low[v], dfs[u]\}$

← Nachbar  $u$  vorher schon gesehen → Rückkante

# Algorithmus → Implementierung

## Anmerkung zur Umsetzung in (Pseudo)code

- zwei Varianten
  - rekursiv: jeder Knoten ruft DFS für alle seine Kinder auf
  - iterativ: ohne rekursive Aufrufe

## Rekursive Implementierung

- schön einfach
- backtracking passiert quasi automatisch
- Problem: möglicherweise tiefe Rekursion → verbraucht viel Stack-Speicher
- Stack-Speicher ist statisch alloziert → wenn der voll ist bricht das Programm ab  
(stack overflow)

## Iterative Implementierung

- umständlich, wenn man beim Backtracken etwas tun möchte
- man muss sich dann im Prinzip den Rekursionsbaum manuell mitverwalten
- Vorteil: verbraucht hauptsächlich (dynamisch allozierten) Heap-Speicher

# Zusammenfassung & Ausblick

## DFS – Tiefensuche

- sehr grundlegender Algorithmus zur Graphtraversierung
- Basis für das Lösen vieler verschiedener Probleme
- zwei wichtige Zeitpunkte:
  - entdecken eines Knotens
  - backtracken von einem Kind zum Elter

## Ein bisschen Graphentheorie

- Brücken in Graphen
- strukturelle Einsicht → effizienter Algorithmus

## Nächstes Mal

- mehr DFS auf gerichteten Graphen
- Wie zieht man sich richtig an?