Zweizusammenhang und starker Zusammenhang

Carsten Gutwenger

Vorlesung

Algorithmen und Datenstrukturen

WS 08/09 • 15. Januar 2009

technische universität dortmund

department of computer science

Zweizusammenhang

Betrachte ein Netzwerk (Graph)

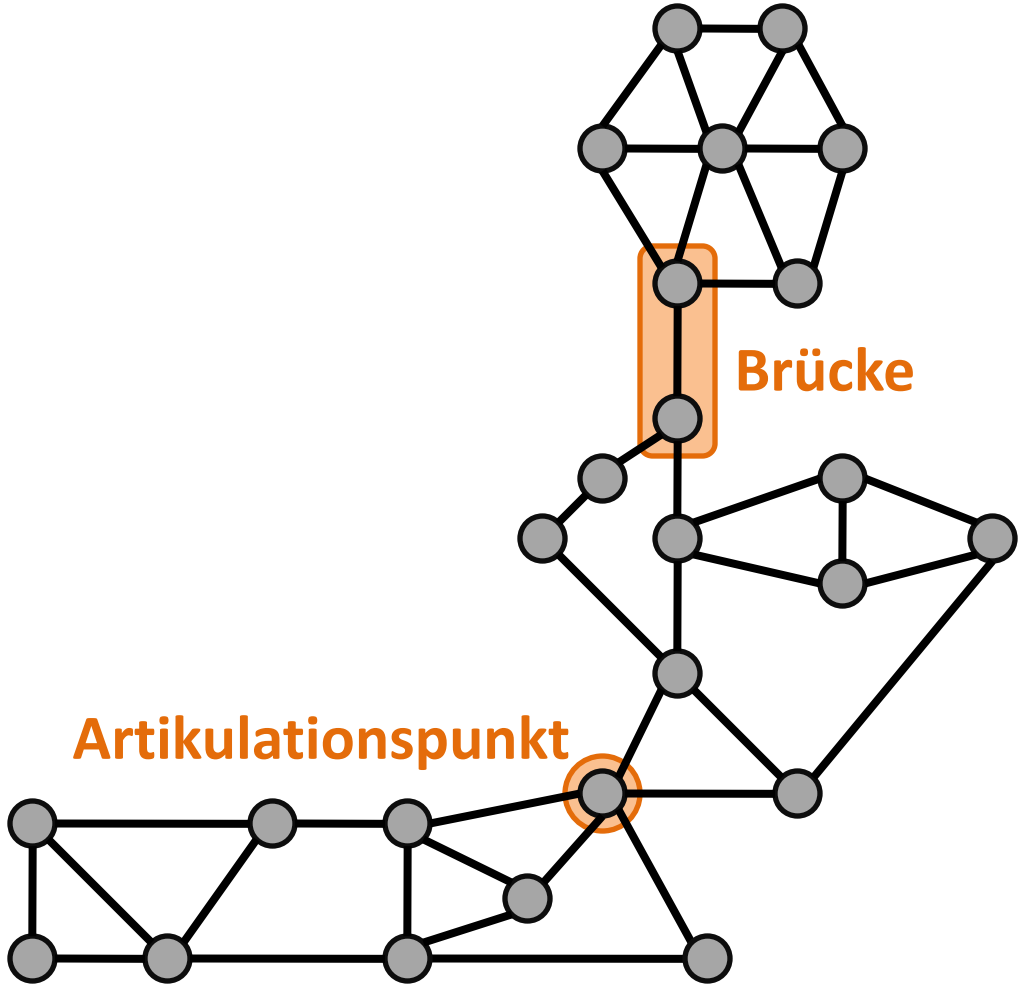
- Z.B. Computernetzwerk, Flug- oder Schienennetzwerk

Einfacher Zusammenhang:

- Es existiert *eine* Verbindung zwischen *je zwei* Punkten

Was geschieht, wenn eine Verbindung oder ein Knoten im Netzwerk ausfällt?

Beispiel: Brücke, Artikulation



Brücke, Artikulation

Sei $G=(V,E)$ ein zusammenhängender Graph.

Definition: Eine Kante $e \in E$ heißt *Brücke* gdw. $G-e$ nicht zusammenhängend ist.

Definition: Ein Knoten $v \in V$ heißt *Artikulation (Schnittknoten, cut vertex)* gdw. $G-v$ nicht zusammenhängend ist.

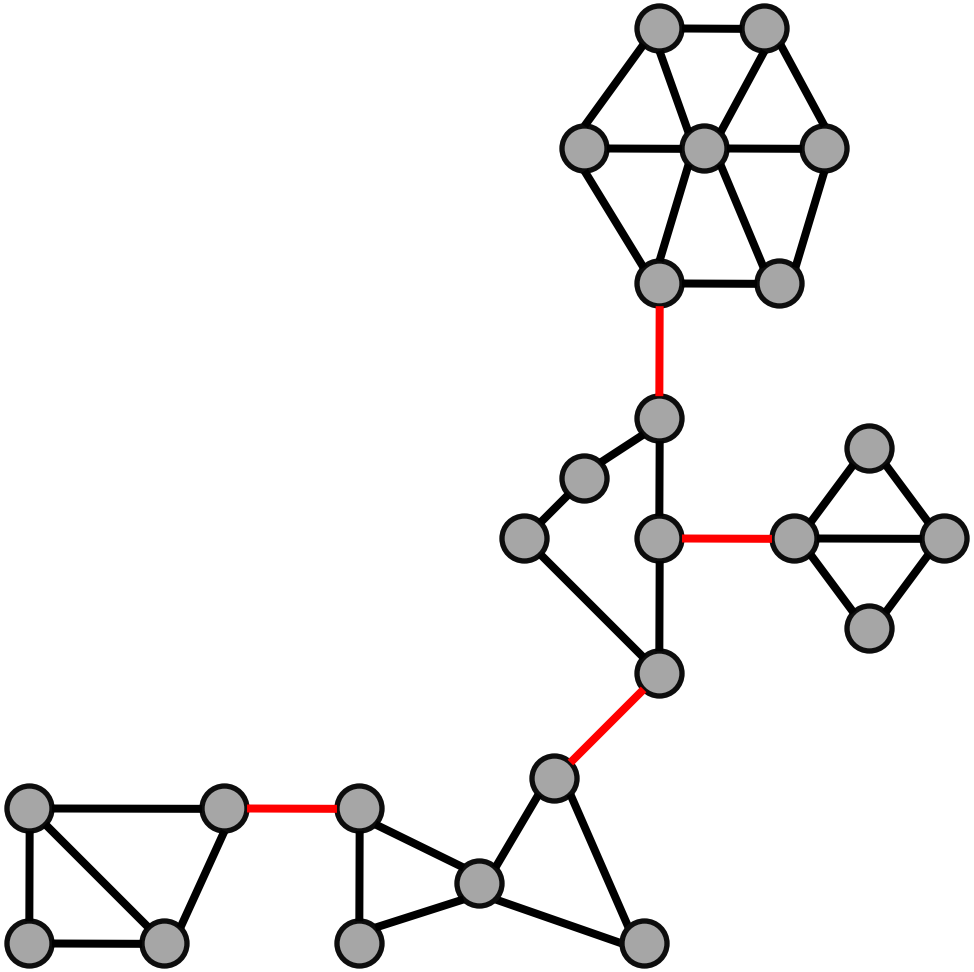
Kanten-zusammenhängend

Sei $G=(V,E)$ ein zusammenhängender Graph

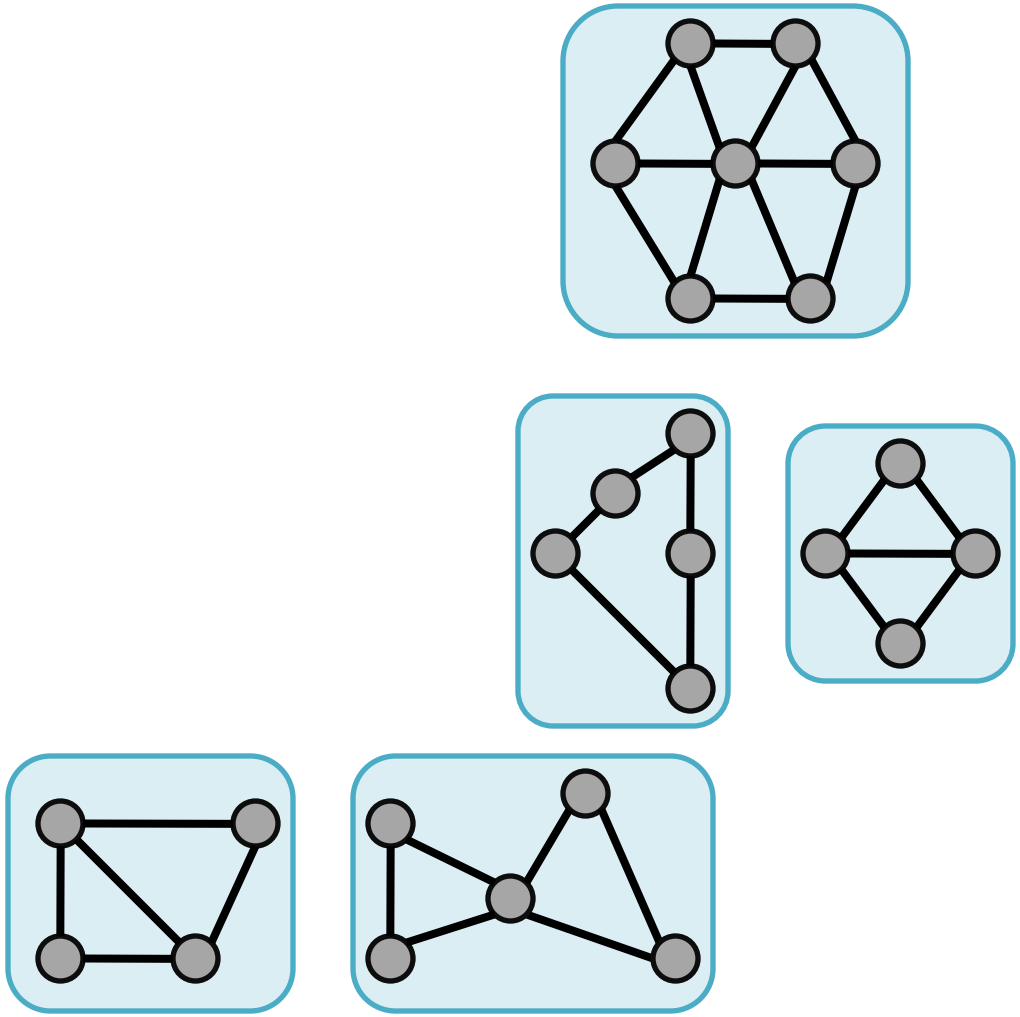
Definition: Enthält G keine Brücke, dann heißt G *kanten-zusammenhängend (edge-connected)*, sonst *kanten-separierbar (edge-separable)*.

Löschen aller Brücken in G $\Rightarrow$ *Brückenzusammenhangskomponenten (bridge-connected components)*

Beispiel: Brücken-ZKen



Beispiel: Brücken-ZKen



2-zusammenhängend

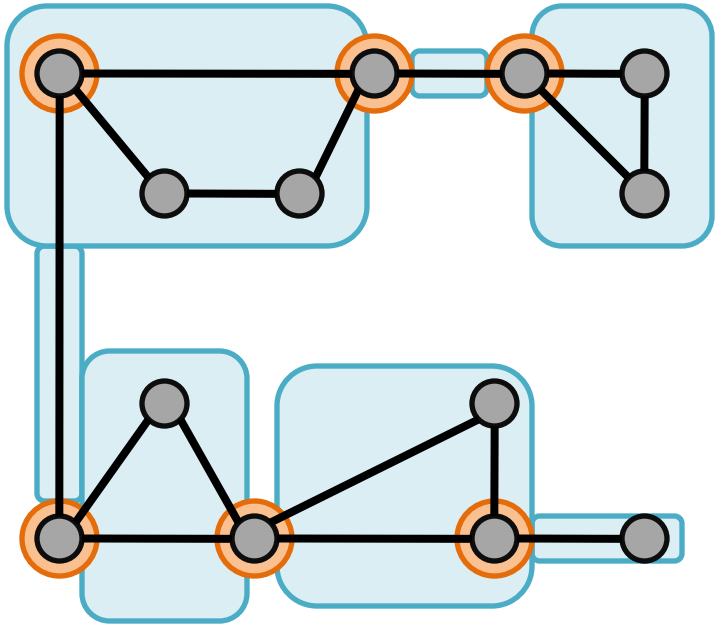
Sei $G=(V,E)$ ein zusammenhängender Graph mit $|V| \geq 3$

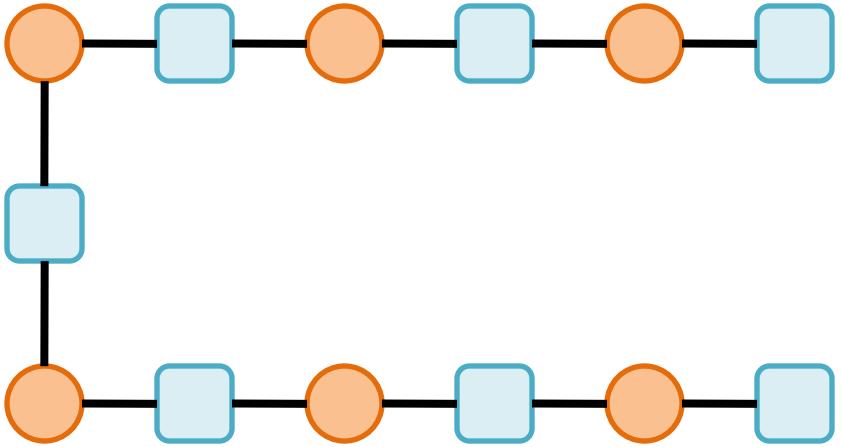
Definition: Enthält G keine Artikulation, dann heißt G *2-zusammenhängend (biconnected)*.

Äquivalent: Jedes Paar von Knoten ist durch 2 unabhängige Pfade verbunden.

Definition: Die *Zweizusammenhangskomponenten (biconnected components)* von G sind seine (Kanten-) maximalen 2-zusammenhängenden Teilgraphen.

Beispiel: 2-Zusammenhangskomponenten





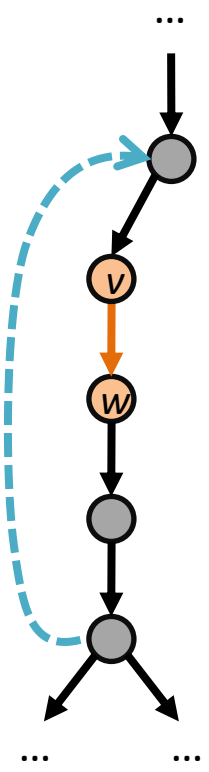
Block-Cutvertex-Tree

Test auf Brücken

Idee: Verwende DFS-Baum

- Baumkanten
- Rückwärtskanten $\Rightarrow$ können keine Brücken sein!

Eigenschaft:
Eine Baumkante $v \rightarrow w$ ist eine Brücke **gdw.** es keine Rückwärtskante gibt, die einen Nachfolger von w mit einem Vorgänger von w verbindet.



low-Werte

Sei $num[v]$ die DFS-Nummer von Knoten v

$low[v] = \min \{ num[v] \} \cup \{ num[x] \mid v \rightarrow^* - x \}$

0 oder mehr Baumkanten und eine Rückwärtskante

Kann leicht bei DFS-Traversierung berechnet werden:

- Initialisierung mit $num[v]$ beim ersten Besuch von v
- Update für Rückwärtskanten (v,w), falls $num[w] < low[v]$
- Update bei Rückkehr von der Rekursion für Kind w, falls $low[w] < low[v]$

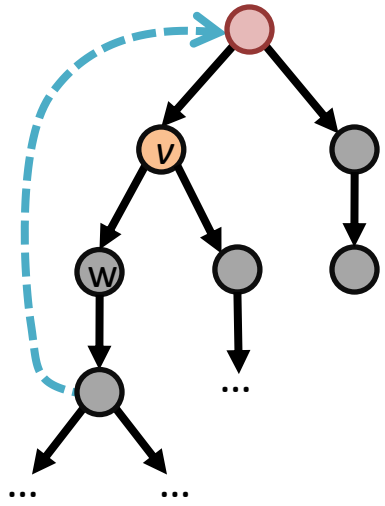
$\Rightarrow$ Brücken können mit DFS in Linearzeit gefunden werden!

Test auf 2-Zusammenhang (1)

Eigenschaft:

Ein Knoten *v* ist Artikulation, gdw.

- *v* ist Wurzel und hat mindestens 2 Kinder; oder
- *v* ist nicht Wurzel und $low[w] \geq num[v]$ für ein Kind *w* von *v*



Test auf 2-Zusammenhang (2)

- **Gegeben:** Graph $G=(V,E)$
- $num[v]$:= -1 für alle $v \in V$
- $low[v]$
- $nCount$:= 0 (Vergabe von DFS-Nummern)
- **Aufruf:** *cutvertex* = dfsBicon(*G*.firstNode(),0)

Test auf 2-Zusammenhang (3)

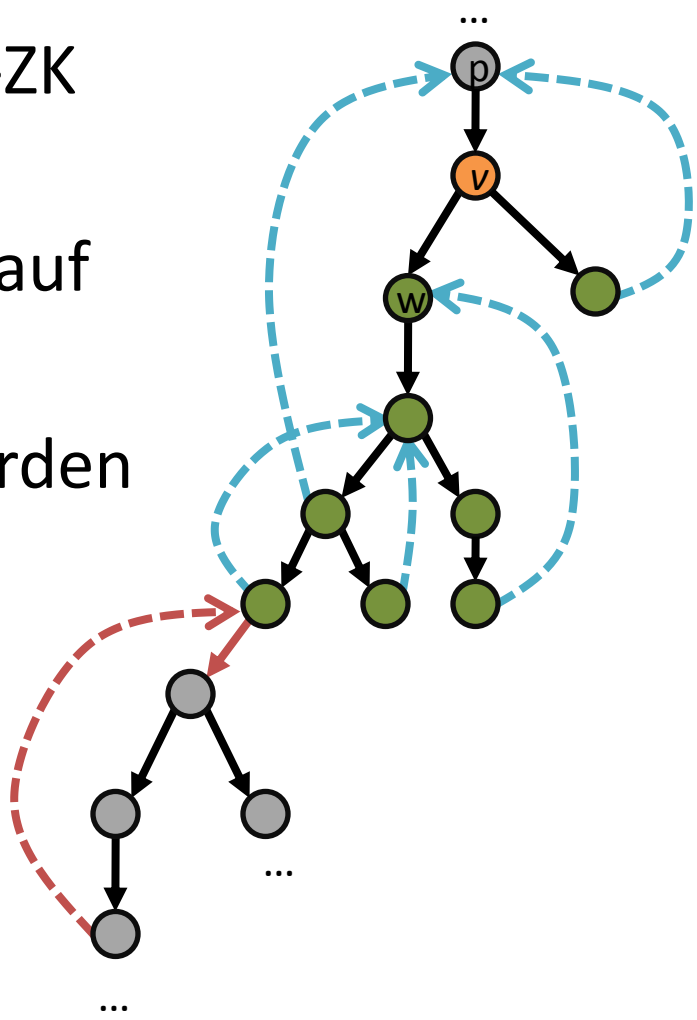
```
function DfsBicon(v, parent) : node
    low[v] := num[v] := nCount++
    firstChild := 0
    forall e=(v,w)∈E do
        if num[w] = -1 then // Baumkante
            if firstChild = 0 then firstChild := w
            cutVertex := dfsBicon(w,v)
            if cutVertex ≠ 0 then return cutVertex
            if low[w] ≥ num[v] and (w ≠ firstChild or parent ≠ 0) then
                return v
            low[v] := min(low[v], low[w])
        else if w ≠ parent then // Rückwärtskante
            low[v] := min(low[v], num[w])
    return 0
```

Test auf Artikulation

Update von low-Wert

2-Zusammenhangskomponenten (1)

- Bestimme für jede Kante *e* ihre 2-ZK *comp*[*e*] (*nComp* := 0)
- Legen Knoten bei erstem Besuch auf Stack *S*
- Beim Identifizieren einer 2-ZK werden Knoten von *S* genommen



2-Zusammenhangskomponenten (2)

```
function DfsBicon(v, parent) : node
    low[v] := num[v] := nCount++
    firstChild := 0
    forall e=(v,w)∈E do
        if num[w] = -1 then // Baumkante
            if firstChild = 0 then firstChild := w
            cutVertex := dfsBicon(w,v)
            if cutVertex ≠ 0 then return cutVertex
            if low[w] ≥ num[v] and (w ≠ firstChild or parent ≠ 0) then
                return v
            low[v] := min(low[v], low[w])
        else if w ≠ v then // Rückwärtskante
            low[v] := min(low[v], num[w])
    return 0
```

2-Zusammenhangskomponenten (3)

```
function DfsBCC(v, parent)
    low[v] := num[v] := nCount++
    S.push(v)
    forall e=(v,w)∈E do
        if num[w] = -1 then // Baumkante
            dfsBCC(w,v)
            low[v] := min(low[v], low[w])
        else // Rückwärtskante
            low[v] := min(low[v], num[w])
    if parent ≠ 0 and low[v] ≥ num[parent] then
        do w = S.pop()
            comp[e] = nComp für alle e=(w,x) mit num[w] > num[x]
        while w ≠ v
        nComp++
```

Zusammenfassung

Theorem: Mit Hilfe von DFS

- können in Linearzeit die Brücken eines Graphen gefunden werden.
- kann in Linearzeit getestet werden, ob ein Graph 2-zusammenhängend ist.
- können in Linearzeit die 2-Zusammenhangskomponten eines Graphen bestimmt werden.

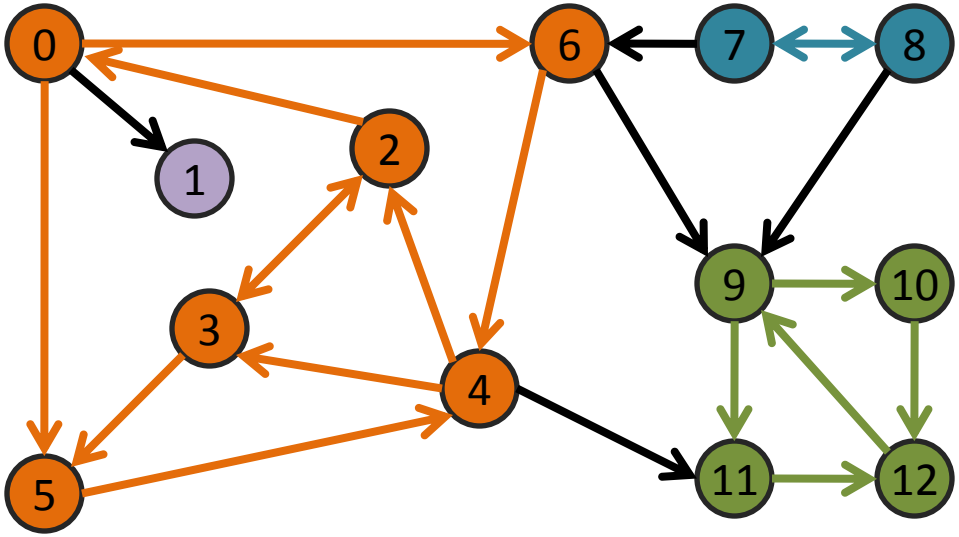
Starke Zusammenhangskomponenten

- in ungerichteten Graphen:
  Ex. Weg von s nach $t \Rightarrow$ ex. Weg von t nach s
- in gerichteten Graphen:
  Ex. Weg von s nach t **und** t nach $s \Rightarrow$ s und t *stark zusammenhängend*

Betrachte im folgenden gerichteten Graphen $G=(V,A)$

Definition: Eine maximale Knotenmenge $S \subseteq V$ mit x und y stark zusammenhängend für alle $x,y \in S$ heißt *starke Zusammenhangskomponente* von G.

Beispiel: SZK



4 starke Zusammenhangskomponenten:

- 0, 2, 3, 4, 5, 6
- 1
- 7, 8
- 9, 10, 11, 12

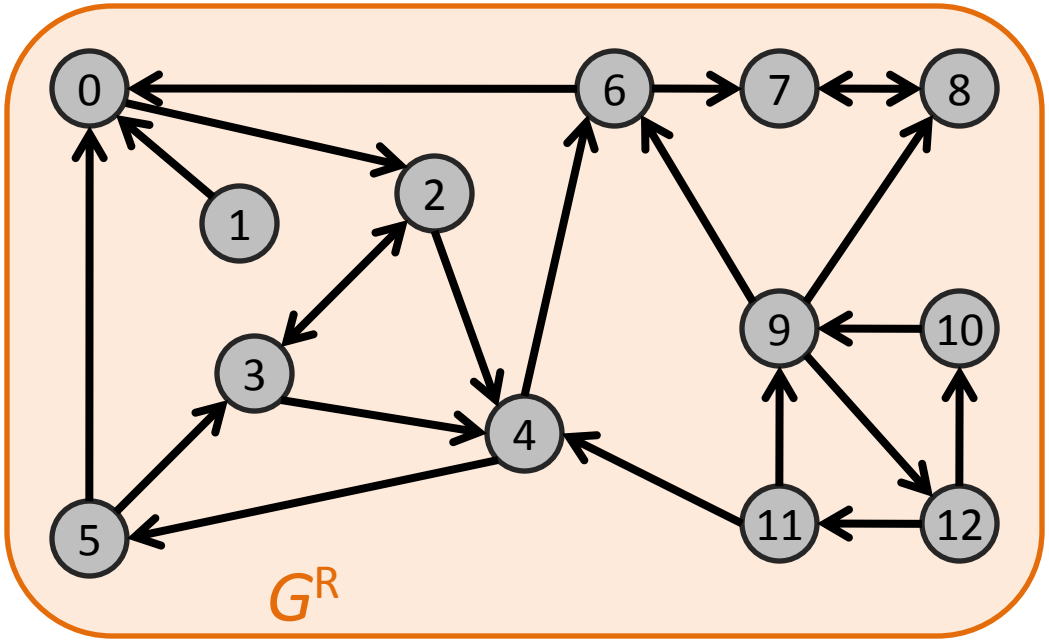
Algorithmus von Kosaraju

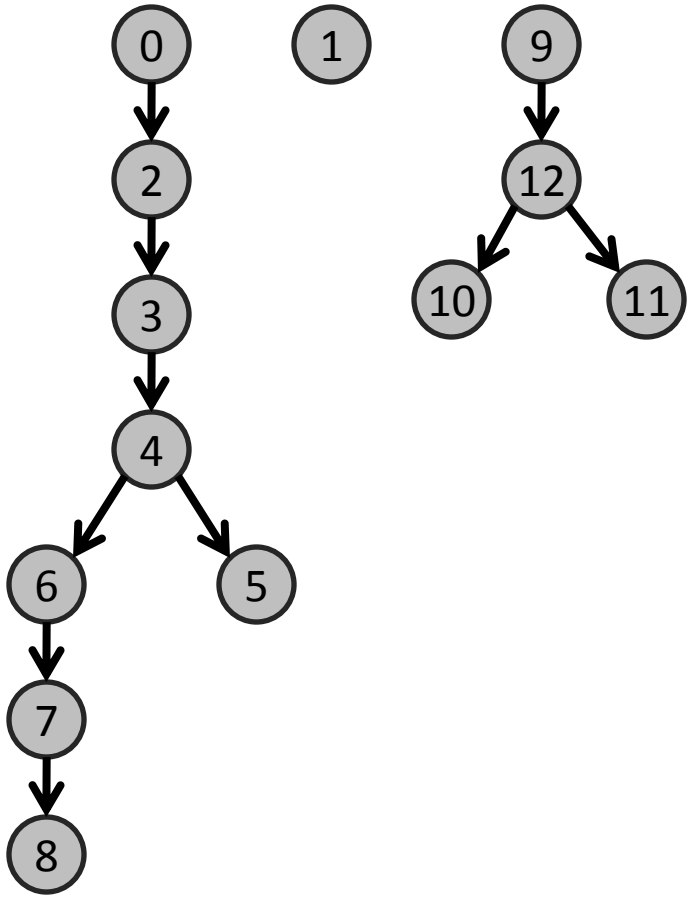
Definition: Der *reverse Graph* $G^R=(V,A^R)$ von $G=(V,A)$ ist gegeben durch $A^R=\{ (w,v) \mid (v,w)\in A \}$.

Algorithmus:

- Führe DFS auf G^R aus
- Seien $v_1,\ldots,v_n$ die Knoten in Postorder-Reihenfolge (d.h. gemäß *completion number*)
- Führe DFS so auf G aus, dass die Knoten in der Reihenfolge $v_n,\ldots,v_1$ betrachtet werden.
- Dann definieren die Bäume im berechneten DFS-Wald die SZKen von G

Beispiel: Kosaraju (1)

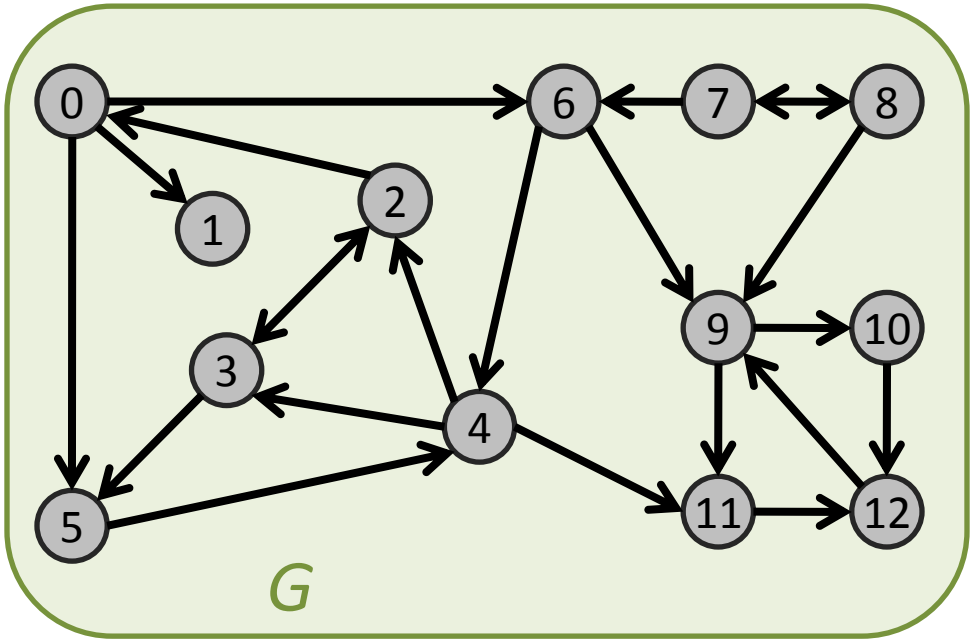


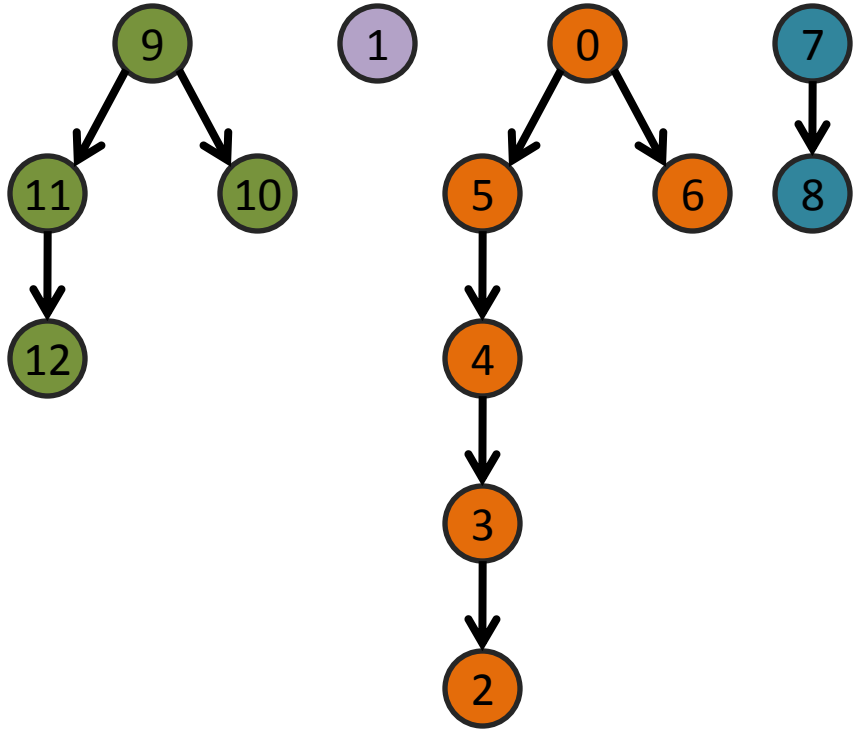


Postorder:

8, 7, 6, 5, 4, 3, 2, 0, 1, 10, 11, 12, 9

Beispiel: Kosaraju (2)

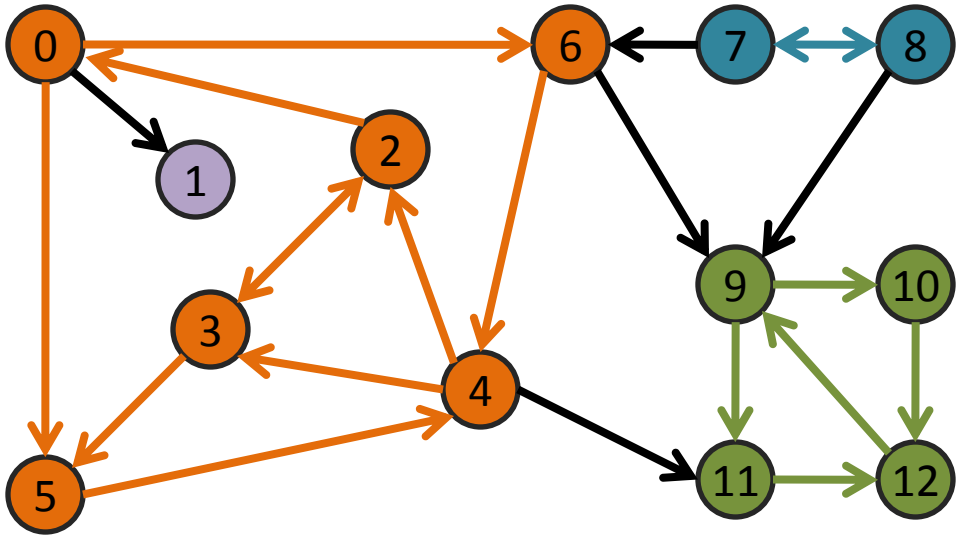




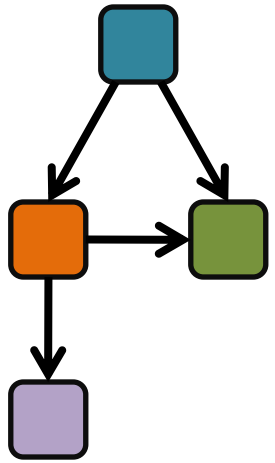
Postorder:

8, 7, 6, 5, 4, 3, 2, 0, 1, 10, 11, 12, 9

Beispiel: Ergebnis



Graph *G* mit SZKen



Struktur der SZKen

Korrektheit (1)

Zu Zeigen: *s* und *t* stark zusammenhängend $\Leftrightarrow$ *s* und *t* im selben DFS-Baum

***s* und *t* stark zusammenhängend:**

- Wenn der erste besucht wird, gibt es einen Pfad zum zweiten $\Rightarrow$ beide im selben Baum

Korrektheit (2)

s und t im selben DFS-Baum:

- Sei r Wurzel dieses Baums $\Rightarrow r \rightarrow^* s$ in G
- $\Rightarrow s \rightarrow^* r$ in G^R **und** postorder(r) > postorder(s)
- Beh.: Es gilt auch $r \rightarrow^* s$ in G^R
- Falls nicht: $\Rightarrow$ postorder(s) > postorder(r) **Widerspruch!**
- analog: $r \rightarrow^* t$ in G^R
- insgesamt: $s \leftrightarrow^* r \leftrightarrow^* t$ in G
  $\Rightarrow s$ und t stark zusammenhängend

Zusammenfassung

Theorem: Der Algorithmus von Kosaraju findet die starken Zusammenhangskomponenten eines gerichteten Graphen (mit Hilfe von DFS) in Linearzeit.