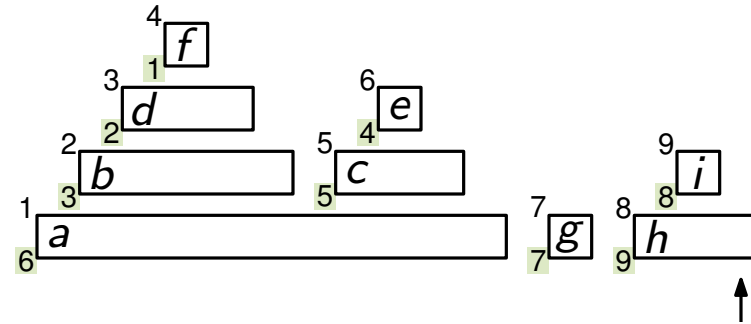
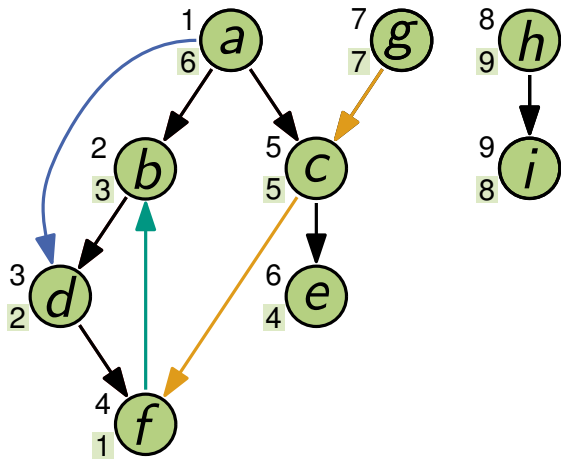


Algorithmen 1

Übung 6 starker Zusammenhang mit DFS, strukturelle Induktion



Tiefensuche (DFS)



Legende:

- ungesehen
- gesehen
- fertig
- ↖ Rückkante
- ↗ Vorkante
- ↘ Querkante

DFS hat viel (nützliche) Struktur

Bisherige Erkenntnisse:

- verschiedene Kantentypen
 - z.B. kreisfrei ⇔ keine Rückkanten
- Topologische Sortierung mittels FIN
- $low(v)$ für Brücken, cut-vertices

	DFS-Nummer	FIN-Nummer
Rückkante	groß → klein	klein → groß
Vorkante	klein → groß	groß → klein
Querkante	groß → klein	groß → klein
Baumkante	klein → groß	groß → klein

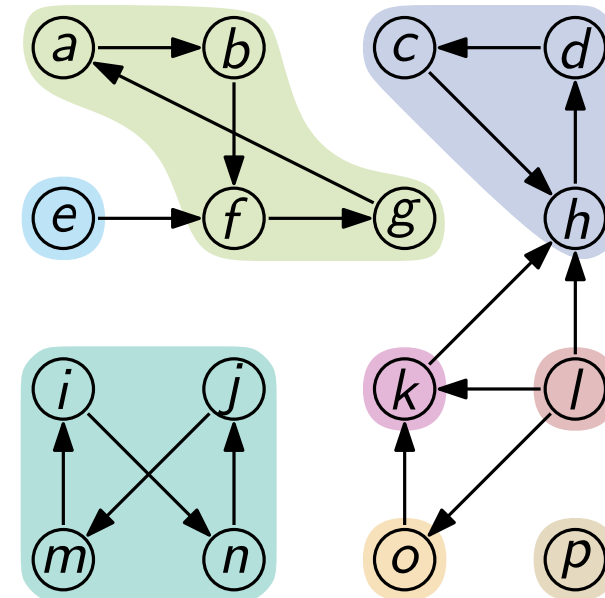
Zusammenhang

Ungerichtete Graphen

- v erreicht w : \exists Pfad von v nach w
 - Äquivalenzrelation
 - Äquivalenzklassen: *Zusammenhangskomponenten*

Gerichtete Graphen

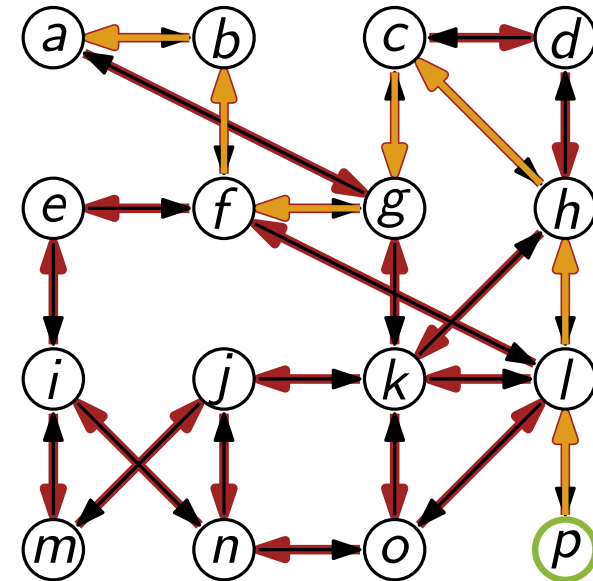
- Erreichbarkeit nicht mehr symmetrisch
 - *starker Zusammenhang*
 - Pfad in beide Richtungen
 - *schwacher Zusammenhang*
 - Pfad in zugrundeliegendem ungerichteten Graphen
- } Jetzt gleich
- } Einfach zu Berechnen



SCCs – Berechnung

Einfacher Ansatz

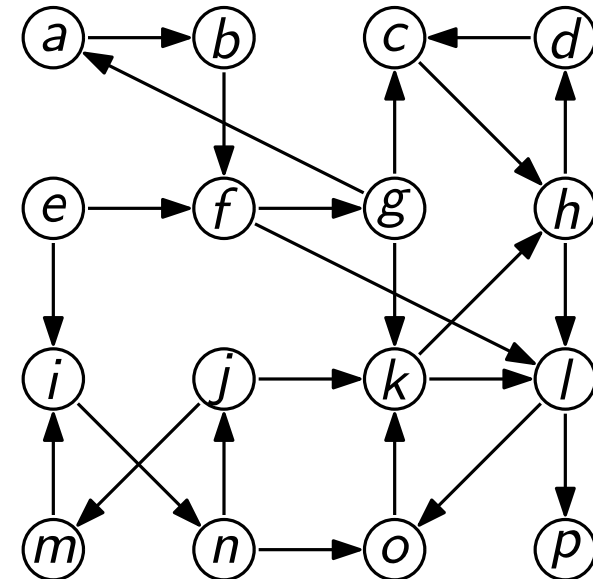
- Für jeden Knoten Zusammenhangskomponente berechnen
 - $\text{reach}_G(v)$: von v erreichbare Knoten in G
 - $\text{scc}(v) = \text{reach}_G(v) \cap \text{reach}_{\text{rev}(G)}(v)$
 - G mit umgekehrten Kantenrichtungen
 - $a \in \text{reach}_{\text{rev}(G)}(p)$
 - p von a erreichbar in G



SCCs – Berechnung

Einfacher Ansatz

- Für jeden Knoten Zusammenhangskomponente berechnen
 - $\text{reach}_G(v)$: von v erreichbare Knoten in G
 - $\text{scc}(v) = \underbrace{\text{reach}_G(v) \cap \text{reach}_{\text{rev}(G)}(v)}_{\substack{\text{von } v \text{ erreichbar} \\ \text{und erreicht } v}}$



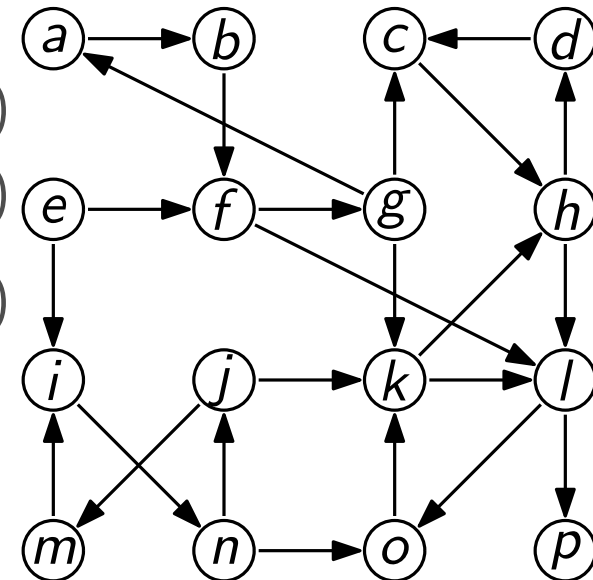
SCCs – Berechnung

Einfacher Ansatz

- Für jeden Knoten Zusammenhangskomponente berechnen
 - $\text{reach}_G(v)$: von v erreichbare Knoten in G $\mathcal{O}(|V| + |E|)$
 - $\text{scc}(v) = \text{reach}_G(v) \cap \text{reach}_{\text{rev}(G)}(v)$ $\mathcal{O}(|V| + |E|)$
- Berechne so $\text{scc}(v)$ für alle $v \in V$ $\mathcal{O}(|V| \cdot (|V| + |E|))$

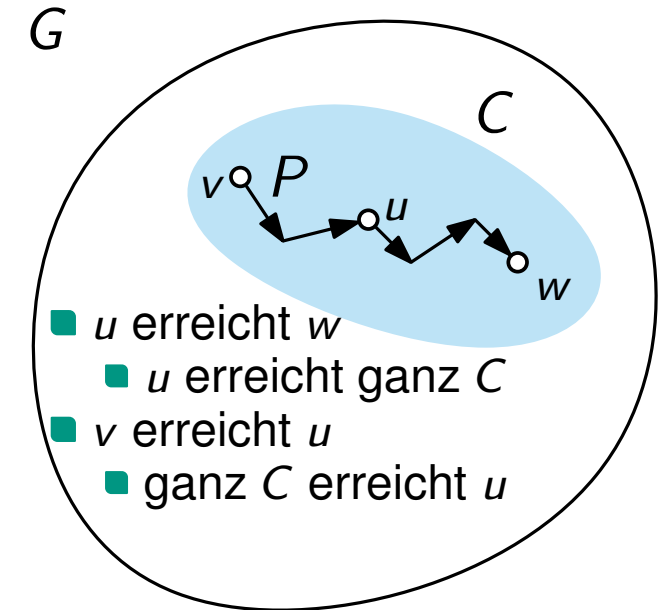
Geht das schneller?

- nur eine Zusammenhangskomponente
 - $|V|$ Zusammenhangskomponenten
 - Allgemeiner Fall: auch in $\mathcal{O}(|V| + |E|)$
 - Technik: DFS!
- } $\mathcal{O}(|V| + |E|)$



SCCs – Struktur

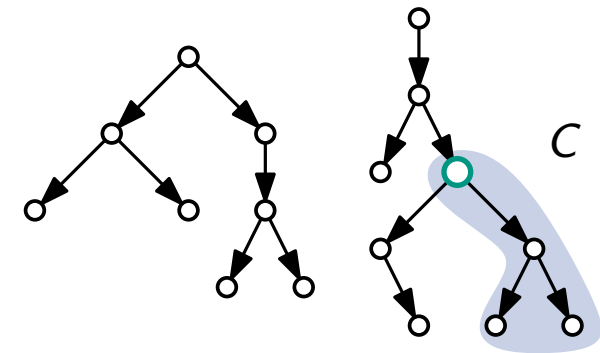
Beobachtung: Seien v, w Knoten in der gleichen SCC C und P ein Pfad von v nach w . Dann liegt P auch in C .



SCCs – Struktur

Beobachtung: Seien v, w Knoten in der gleichen SCC C und P ein Pfad von v nach w . Dann liegt P auch in C .

Lemma: Betrachte einen DFS-Wald F eines gerichteten Graphen G . Jede SCC C von G enthält genau einen Knoten w mit $\text{parent}(w) \notin C$ oder $\text{parent}(w) = \perp$.



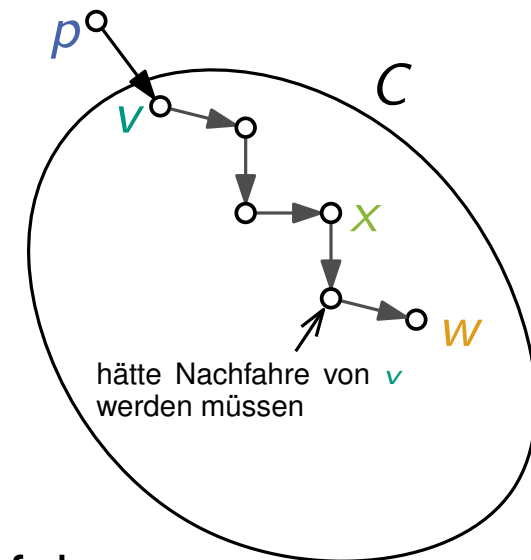
SCCs – Struktur

Beobachtung: Seien v, w Knoten in der gleichen SCC C und P ein Pfad von v nach w . Dann liegt P auch in C .

Lemma: Betrachte einen DFS-Wald F eines gerichteten Graphen G . Jede SCC C von G enthält genau einen Knoten w mit $\text{parent}(w) \notin C$ oder $\text{parent}(w) = \perp$.

Beweis:

- sei v in C mit kleinster DFS-Nummer
 - falls v Elternknoten p hat: $p.\text{DFS} < v.\text{DFS}$ also $p \notin C$
- betrachte nun anderen Knoten $w \in C$
 - w Nachfahre von v in DFS Baum
 - Andernfalls: betrachte v - w -Pfad P , sei x hinterster Nachfahre von v



SCCs – Struktur

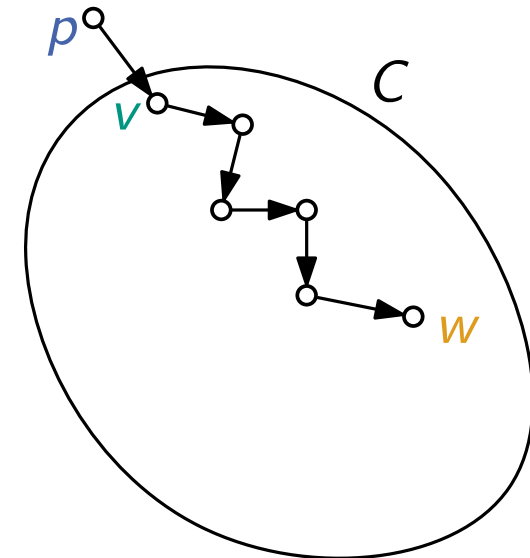
Beobachtung: Seien v, w Knoten in der gleichen SCC C und P ein Pfad von v nach w . Dann liegt P auch in C .

Lemma: Betrachte einen DFS-Wald F eines gerichteten Graphen G . Jede SCC C von G enthält genau einen Knoten w mit $\text{parent}(w) \notin C$ oder $\text{parent}(w) = \perp$.

Beweis:

- sei v in C mit kleinster DFS-Nummer
 - falls v Elternknoten p hat: $p.\text{DFS} < v.\text{DFS}$ also $p \notin C$
- betrachte nun anderen Knoten $w \in C$
 - w Nachfahre von v in DFS Baum
 - gibt Pfad aus Baumkanten von v zu w
 - nach Beobachtung: Elternknoten von w in C

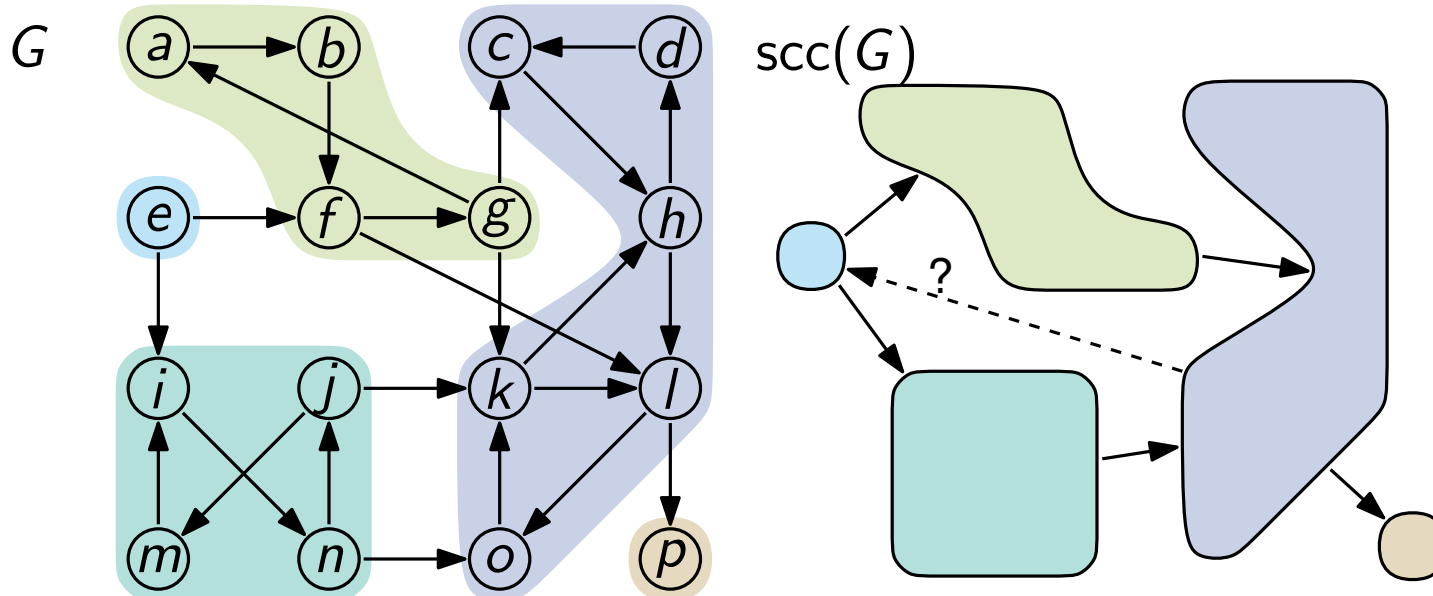
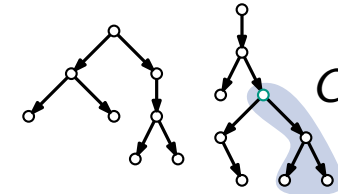
□



SCCs – Mehr Struktur

Lemma: Betrachte einen DFS-Wald F eines gerichteten Graphen G . Jede SCC C von G enthält genau einen Knoten w mit $\text{parent}(w) \notin C$ oder $\text{parent}(w) = \perp$.

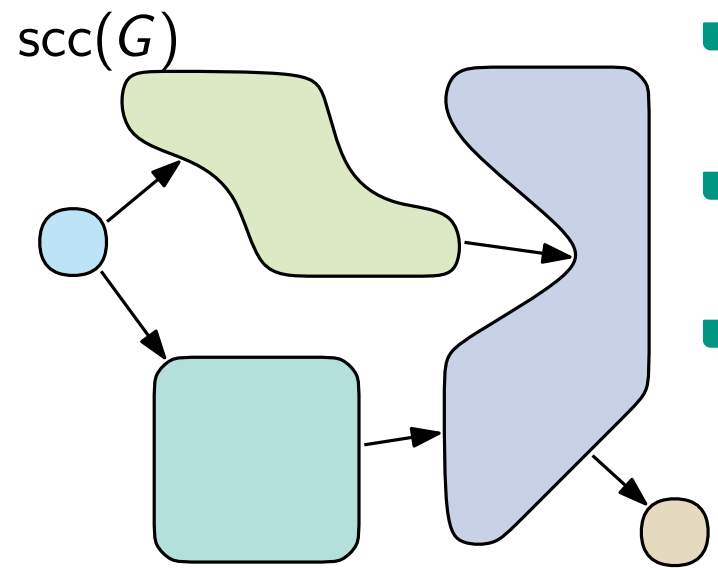
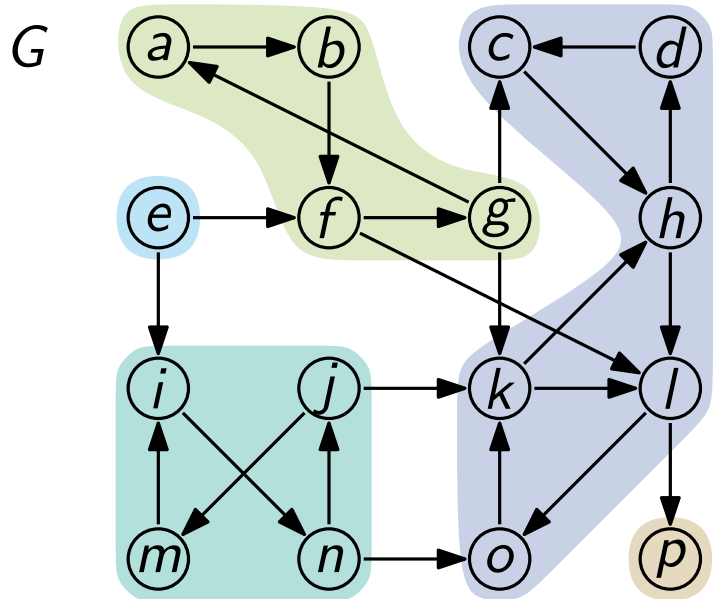
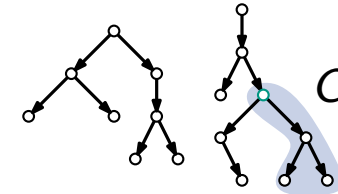
Beobachtung: Die SCCs eines Graphen bilden einen azyklischen Graphen.



SCCs – Mehr Struktur

Lemma: Betrachte einen DFS-Wald F eines gerichteten Graphen G . Jede SCC C von G enthält genau einen Knoten w mit $\text{parent}(w) \notin C$ oder $\text{parent}(w) = \perp$.

Beobachtung: Die SCCs eines Graphen bilden einen azyklischen Graphen.

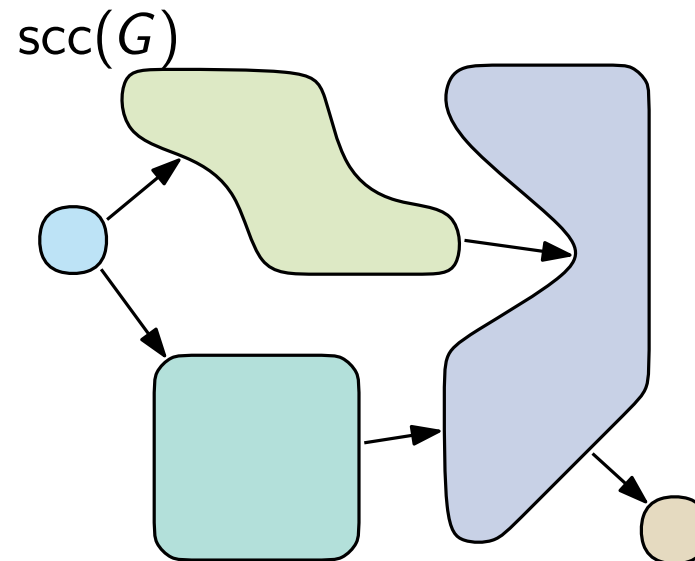
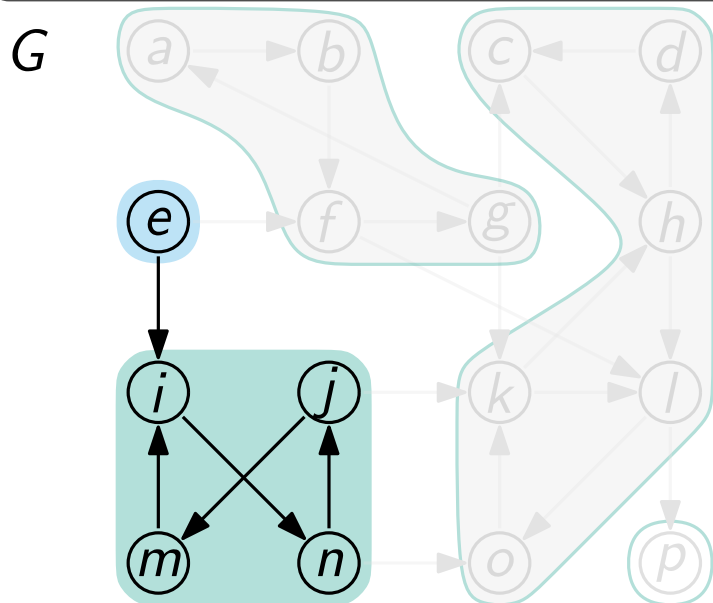
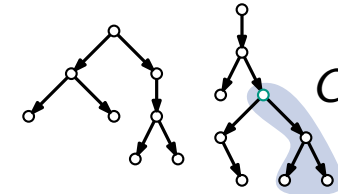


- Senken-Komponente: Senke in $\text{scc}(G)$
- SCC von Knoten v in Senken-Komp. finden ist einfach
- Algorithmus:
 - solange $V \neq \emptyset$: finde Knoten v in Senken-Komponente
 - finde SCC C von v , lösche C

SCCs – Mehr Struktur

Lemma: Betrachte einen DFS-Wald F eines gerichteten Graphen G . Jede SCC C von G enthält genau einen Knoten w mit $\text{parent}(w) \notin C$ oder $\text{parent}(w) = \perp$.

Beobachtung: Die SCCs eines Graphen bilden einen azyklischen Graphen.



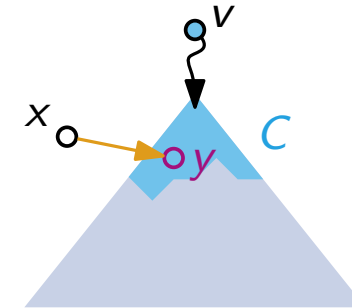
- Senken-Komponente: Senke in $\text{scc}(G)$
- SCC von Knoten v in Senken-Komp. finden ist einfach
- Algorithmus:
 - solange $V \neq \emptyset$: finde Knoten v in Senken-Komponente
 - finde SCC C von v , lösche C

SCCs – Knoten in Senken Komponente finden

Lemma: Der Knoten mit der größten FIN-Nummer liegt in einer Quellen-Komponente.

Beweis:

- sei v Knoten mit größter FIN-Nummer
 - v ist Wurzel von zuletzt gefundenem Baum im DFS-Wald
 - v ist einziger Knoten ohne Elter in v 's SCC C
- Angenommen $(x, y) \in E$ mit $x \notin C, y \in C$
 - x nicht im DFS Baum unter v
 - also: (x, y) **Querkante**
 - x nach y gefunden, aber vor v fertig
 - x Nachfahre von v
 - Pfad von v zu x und von x zu v ↯



Querkante	DFS-Nummer	FIN-Nummer
	groß → klein	groß → klein

□

SCCs – Knoten in Senken Komponente finden 2

Lemma: Der Knoten mit der größten FIN-Nummer liegt in einer Quellen-Komponente.

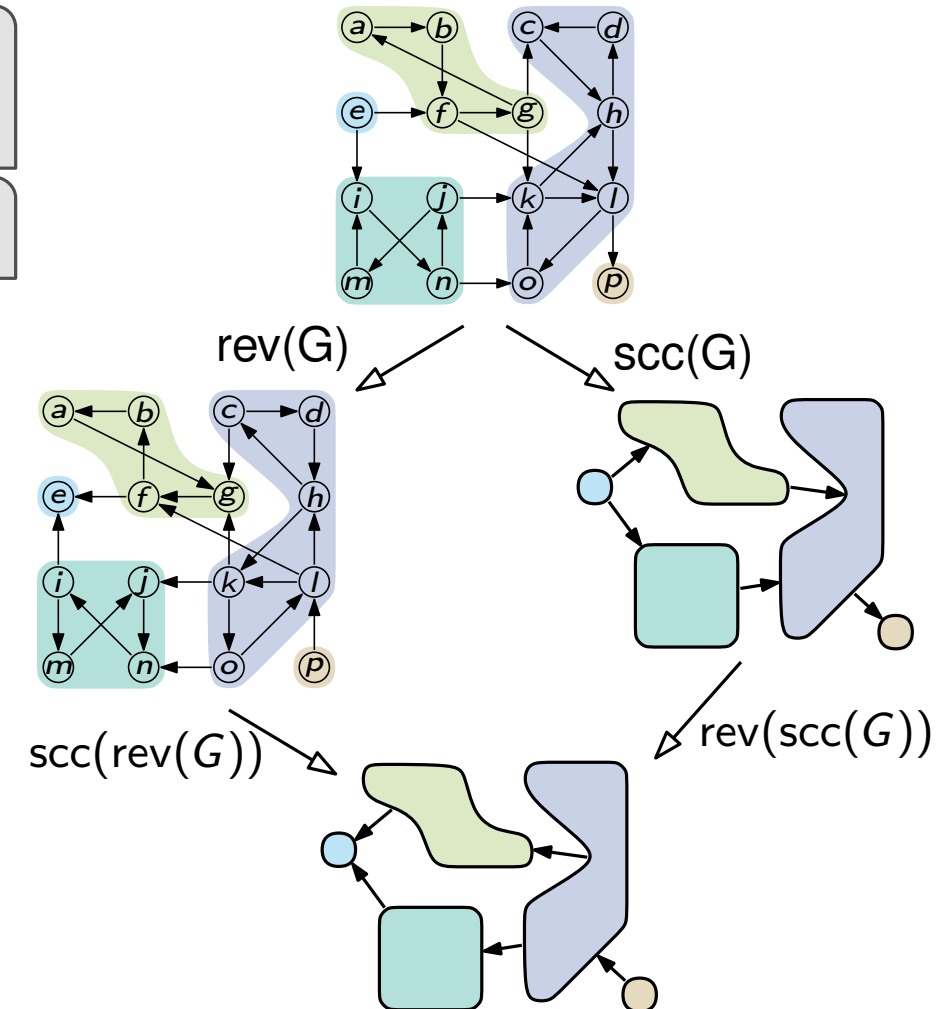
Beobachtung: $\text{rev}(\text{scc}(G)) = \text{scc}(\text{rev}(G))$

Betrachte Knotenpaar $u, v \in V$

- Fall 1: u erreicht v und v erreicht u
- Fall 2: u erreicht v , aber v erreicht u nicht (o.B.d.A.)
- Fall 3: u erreicht v nicht und v erreicht u nicht

Idee:

- DFS Traversierung von $\text{rev}(G)$
- Knoten v mit Größter FIN-Nummer liegt in Quellen-Komp. von $\text{rev}(G)$
- v liegt in Senken-Komponente von G

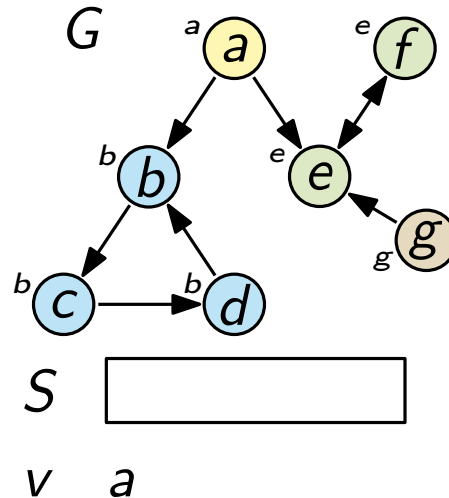


SCCs – Pseudocode

KosarajuSharir(*Graph* $G = (V, E)$)

```

S := new Stack
for Node  $v$  in  $V$  do
  set  $v$  unmarked
   $v.root := \perp$ 
// Erste Tiefensuche
for Node  $v$  in  $V$  do
  if  $v$  is unmarked then
    | revDFS( $G, S, u$ )
// Zweite Tiefensuche
while  $S$  is not empty do
   $v := \text{pop}(S)$ 
  if  $v.root = \perp$  then
    | sccDFS( $G, v, v$ )
  
```



revDFS(*Graph* $G, \text{Stack } S, \text{Node } v$)

```

mark  $v$ 
for Node  $u$  with  $v \in N(u)$  do
  | if  $u$  is unmarked then
    | | revDFS( $G, S, u$ )
  | S.push( $v$ )
  
```

sccDFS(*Graph* $G, \text{Node } v, \text{Node } r$)

```

 $v.root = r$ 
for Node  $u$  in  $N(v)$  do
  | if  $u.root = \perp$  then
    | | sccDFS( $G, u, r$ )
  
```


Evaluierung der Übung



<https://onlineumfrage.kit.edu/evasys/online.php?p=X4APS>

Vollständige Induktion

Summenformel für Quadrate

$$M = \left\{ n \in \mathbb{N} \mid \sum_{i=1}^n (2i - 1) = n^2 \right\}$$

Induktionsanfang $1 \in M$

$$\sum_{i=1}^1 (2i - 1) = 1 = 1^2$$

Induktionsschritt

Angenommen $m \in M$. Dann gilt $m + 1 \in M$:

$$\begin{aligned} \sum_{i=1}^{m+1} (2i - 1) &= 2(m+1) - 1 + \sum_{i=1}^m (2i - 1) = \\ &= 2(m+1) - 1 + m^2 = m^2 + 2m + 1 = (m+1)^2 \end{aligned}$$

Folgerung $M = \mathbb{N}$

Formal

Sei $M \subseteq \mathbb{N}$. Falls gilt

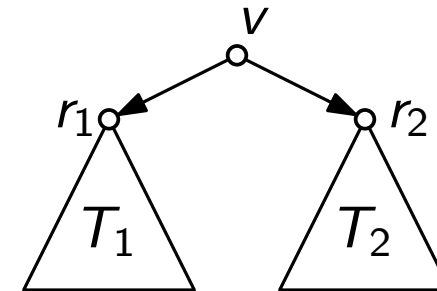
- $1 \in M$; und
 - für alle $m \in M$ ist auch $m + 1 \in M$,
- dann gilt $M = \mathbb{N}$.

Wieso stimmt das?

Binärbäume rekursiv

Definition: Binärbaum

- Graphen mit genau einem Knoten v sind Binärbäume mit Wurzel v
- für disjunkte Binärbäume T_1 und T_2 mit Wurzeln r_1, r_2 und einem weiteren Knoten v , ist $T = (V(T_1) \cup V(T_2) \cup \{v\}, E(T_1) \cup E(T_2) \cup \{(v, r_1), (v, r_2)\})$ ein Binärbaum.



Strukturelle Induktion mit Binärbäumen

Sei M eine Menge von Binärbäumen. Falls gilt

- Binärbäume mit genau einem Knoten sind in M ; und
- für disjunkte Binärbäume T_1 und T_2 in M mit Wurzeln r_1, r_2 und einem weiteren Knoten v , ist $T = (V(T_1) \cup V(T_2) \cup v, E(T_1) \cup E(T_2) \cup \{(v, r_1), (v, r_2)\})$ in M ,

dann ist M die Menge aller Binärbäume.

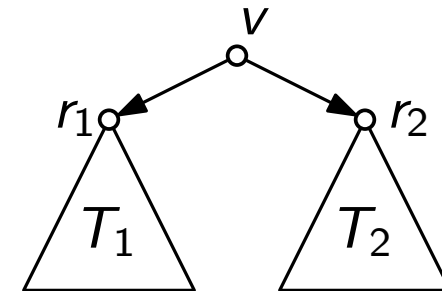
Strukturelle Induktion – Beispiel

Satz über innere Knoten und Blätter

In jedem Binärbaum gibt es einen inneren Knoten (Knoten mit Ausgangsgrad > 0) weniger als Blätter (Knoten mit Ausgangsgrad 0).

Beweis:

- Basisfall: Binärbäume mit einem Knoten
 - Innere Knoten: 0, Blätter: 1
- rekursiver Fall: Wurzel v mit Kindern r_1 und r_2 , darunter Teilbäume T_1, T_2
 - Innere Knoten: $\text{inner}(T_1) + \text{inner}(T_2) + 1$
 - Blätter: $\text{leafs}(T_1) + \text{leafs}(T_2) = \text{inner}(T_1) + \text{inner}(T_2) + 2$
 - Induktionsvoraussetzung: $\text{inner}(T_1) + 1 = \text{leafs}(T_1)$,
 $\text{inner}(T_2) + 1 = \text{leafs}(T_2)$



Strukturelle Induktion – Beispiel 2

Satz über Balance

Sei T ein Binärbaum mit n Knoten, sodass für jeden Teilbaum T' mit Kindbäumen L und R gilt $|h(L) - h(R)| \leq 1$.

Dann gilt $\sqrt{2}^{h(T)-1} \leq n$.

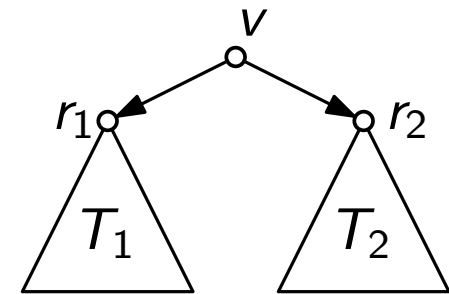
Beweis:

- Basisfall: Binärbäume mit einem Knoten
 - $\sqrt{2}^0 = 1 \leq 1$
- rekursiver Fall: Wurzel v mit Kindbäumen T_1, T_2
 - sei $n_1 = |V(T_1)|, n_2 = |V(T_2)|$; o.B.d.A. $n_1 \leq n_2$
 - dann $\sqrt{2}^{h(T_1)-1} \leq n_1$, zudem: $h(T) \leq h(T_1) + 2$

$$\sqrt{2}^{h(T)-1} \leq \sqrt{2}^{h(T_1)+1} \leq \sqrt{2}^2 \cdot \sqrt{2}^{h(T_1)-1} \leq 2 \cdot n_1 \leq n_1 + n_2 \leq n$$

Frage: Was hat das mit Algorithmik zu tun?

Antwort: Zusammenhang zw. Induktion und rekursiven Algorithmen / DPs



Induktion und Rekursion

Algorithmus für Höhe von Baum (rekursiv)

- Korrektheit:
 - Basisfall: Einzelner Knoten
 - Induktionsschritt: Höhe = 1 + Höhe in Teilbäumen
- Strukturelle Induktion über alle Binärbäume

Topologische Sortierung

- Korrektheit:
 - Induktionsanfang: $|V| = 1$
 - Induktionsschritt: falls **topoSort** für $n - 1$ Knoten korrekt, dann auch für n
- Vollständige Induktion über Anzahl Knoten

height(*Tree T, Node root*)

if *root* is only Node in *T* **then**

return 0

left := **height**(**leftSubtree**(*root*))

right := **height**(**rightSubtree**(*root*))

return 1 + max{*left*, *right*}

topoSort(*acyclic Graph G = (V, E)*)

if $|V| = 1$ **then**

return $\langle V[0] \rangle$

source := **findSource**(**G**)

G' := **removeNode**(*G*, *source*)

return $\langle source \rangle + \langle \mathbf{topoSort}(G') \rangle$