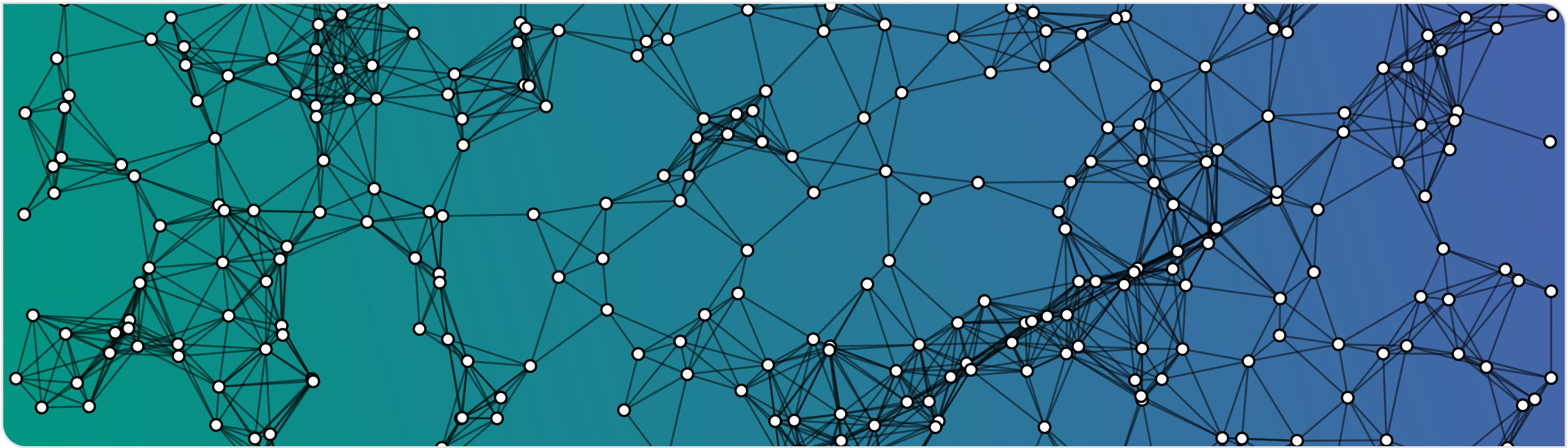
Karlsruher Institut für Technologie

Algorithmen 1

Übung 6
starker Zusammenhang mit DFS, strukturelle Induktion



Ankündigung: Zusatztutorien im August

Idee

- Wiederholung wichtiger Themen
- Üben für Prüfung

Tutorinnen und Tutoren

- Laurin Mayer
- Kilian Wurm
- Joshua Jantzen
- Livien Majer

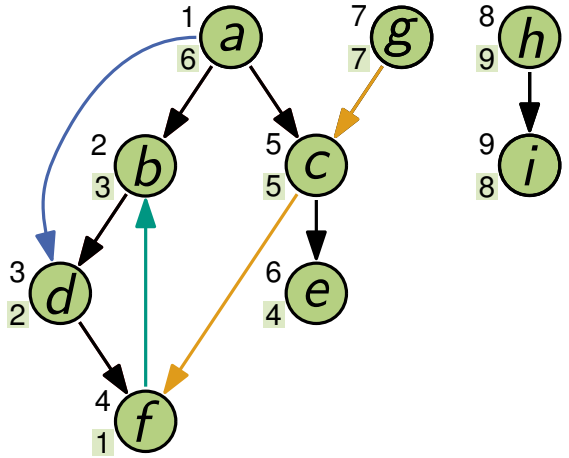
Termine

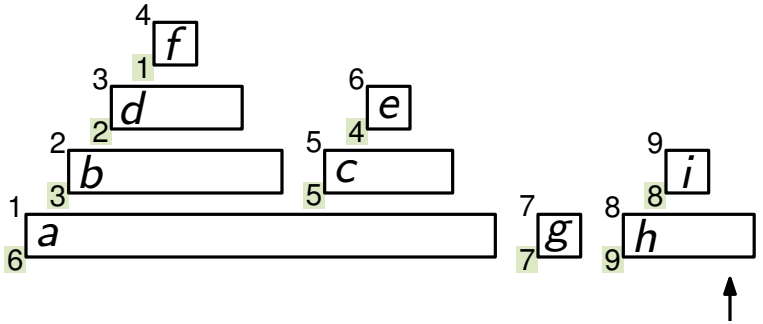
August 2024

	Mo	Di	Mi	Do	Fr	Sa	So
Juli	29	30	31	1	2	3	4
	5	6	7	8	9	10	11
	12	13	14	15	16	17	18
	19	20	21	22	23	24	25
	26	27	28	29	30	31	

- Montags 9:45 und Mittwochs 9:45
- gesamter August

Tiefensuche (DFS)





Legende:

- ungesehen
- gesehen
- fertig

- Rückkante
- Vorkante
- Querkante

DFS hat viel (nützliche) Struktur!

Bisherige Erkenntnisse:

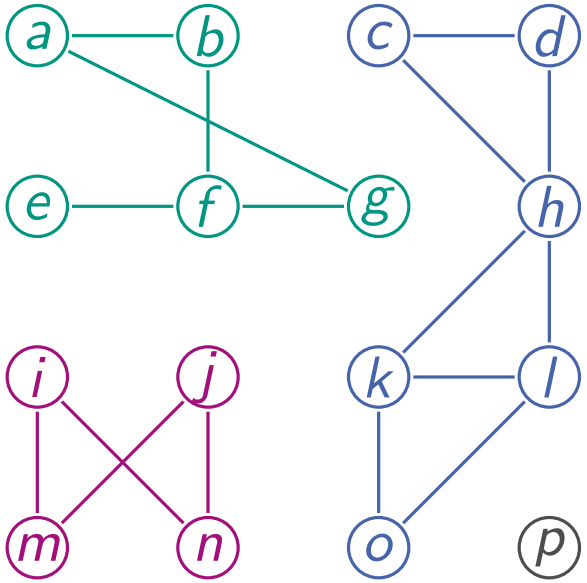
- verschiedene Kantentypen
  - z.B. kreisfrei $\Leftrightarrow$ keine Rückkanten
- Topologische Sortierung mittels FIN
- low(v) für Brücken, cut-vertices

	DFS-Nummer	FIN-Nummer
Rückkante	groß $\rightarrow$ klein	klein $\rightarrow$ groß
Vorkante	klein $\rightarrow$ groß	groß $\rightarrow$ klein
Querkante	groß $\rightarrow$ klein	groß $\rightarrow$ klein
Baumkante	klein $\rightarrow$ groß	groß $\rightarrow$ klein

Zusammenhang

Ungerichtete Graphen

- Def.: v *erreicht* w: $\exists$ Pfad von v nach w
  - Äquivalenzrelation
  - Äquivalenzklassen: *Zusammenhangskomponenten (CC)*



Wie können CCs berechnet werden?

- d.h. für geg. Graph G: ordne jedem $v \in V(G)$ seine CC zu
- DFS von $v \in V(G)$
  - gefundene Knoten gehören zu CC von v
  - falls weiterer unmarkierter Knoten $w \in V(G)$ existiert, wiederhole mit DFS von w

Laufzeit? ($n := |V|$, $m := |E|$, k: Anzahl CCs)

Zusammenhang

Ungerichtete Graphen

- CC: Äquivalenzklassen bezüglich Erreichbarkeit
- Berechnung: BFS/DFS jeweils von markierten Knoten

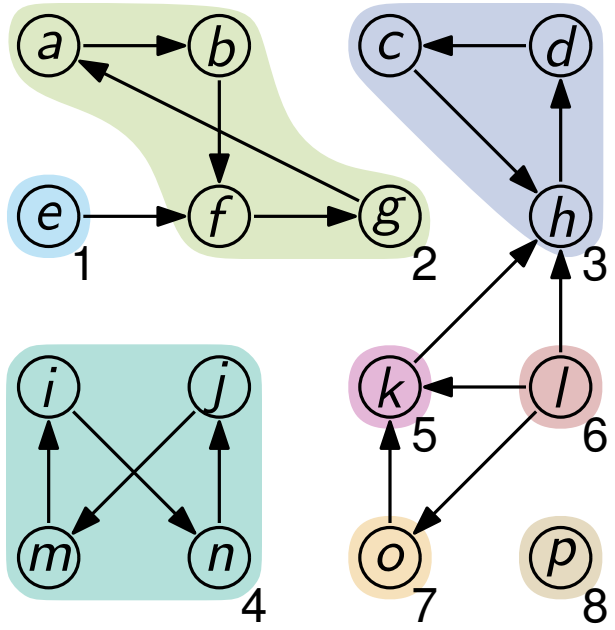
Wie ist das auf gerichtete Graphen?

- Erreichbarkeit nicht mehr symmetrisch
- Gibt es trotzdem sinnvolle Definitionen für CCs?
  - *schwacher Zusammenhang*
    - ignoriere Kantenrichtungen, betrachte G als ungerichtet

    } $\Theta(n+m)$
  - *starker Zusammenhang*
    - Pfad in beide Richtungen
    - $u \sim v$: es gibt uv-Pfad und vu-Pfad

    } Berechnung: Jetzt gleich

Frage: wie viele SCCs hat der Graph?

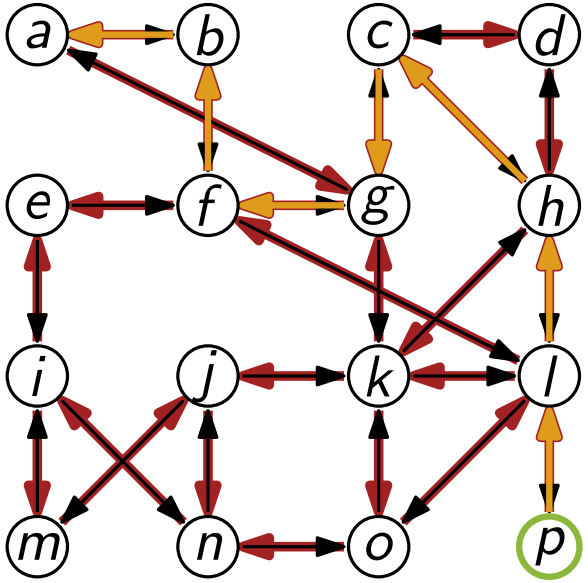


SCCs – Berechnung

Einfacher Ansatz

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

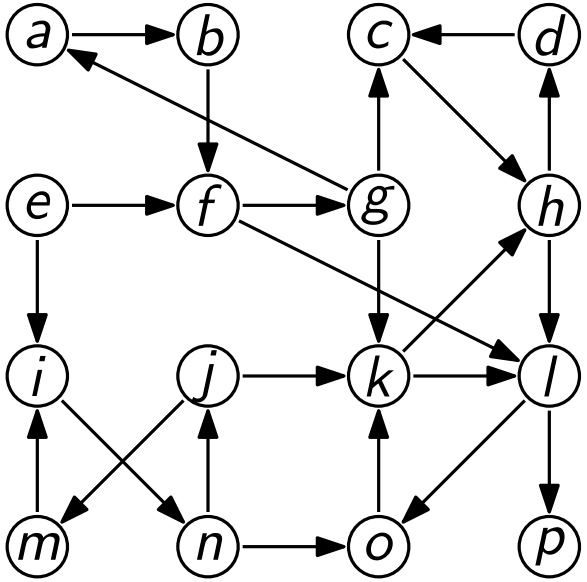
    G mit umgekehrten Kantenrichtungen
    - $a \in \text{reach}_{\text{rev}(G)}(p)$
    - p von a erreichbar in G



SCCs – Berechnung

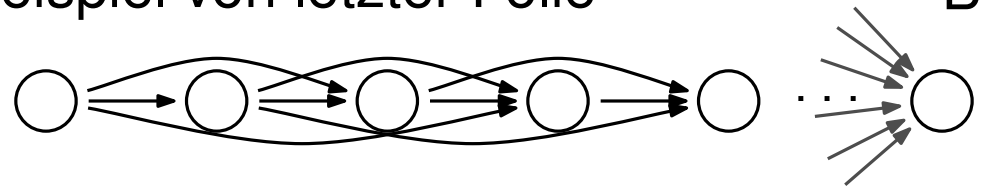
Einfacher Ansatz

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



Idee für Algorithmus

Beispiel von letzter Folie



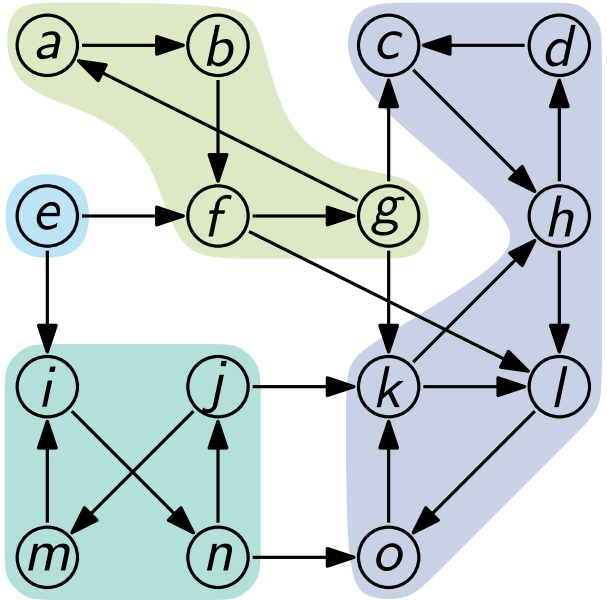
Berechnung von $\text{scc}(v) = \text{reach}_G(v) \cap \text{reach}_{\text{rev}(G)}(v)$?

$\mathcal{O}(1)$

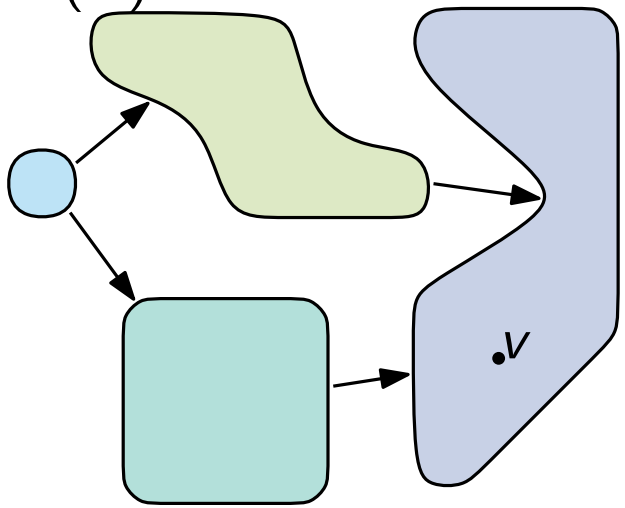
v letzter Knoten in top. Sort.

Was passiert für allgemeine Graphen?

G



$\text{scc}(G)$



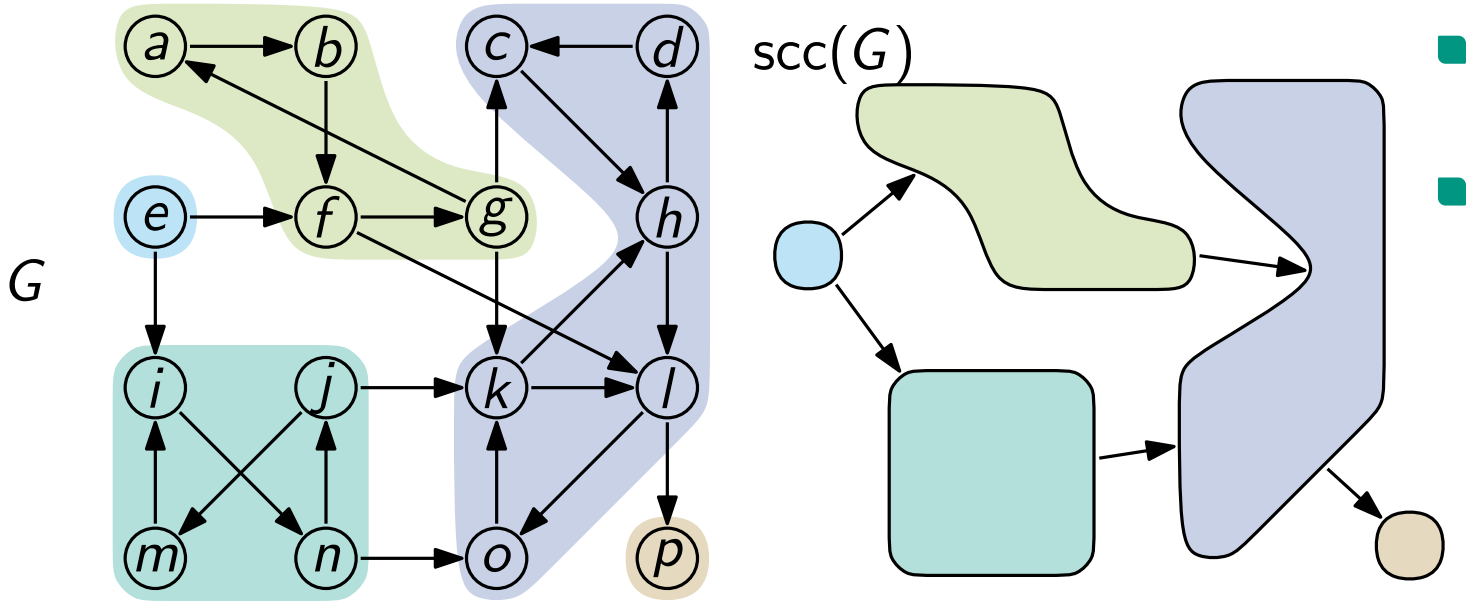
Beobachtung: Die SCCs eines Graphen bilden einen azyklischen Graphen.

- sei v in „Senken-Komponente“
- $\text{scc}(v) = \text{reach}_G(v) \cap \cancel{\text{reach}_{\text{rev}(G)}(v)}$
- Berechnung in $\mathcal{O}(|\text{scc}(v)|)$

Idee für Algo: wiederholt Knoten in Senke finden!

Idee für Algorithmus – Beispiel

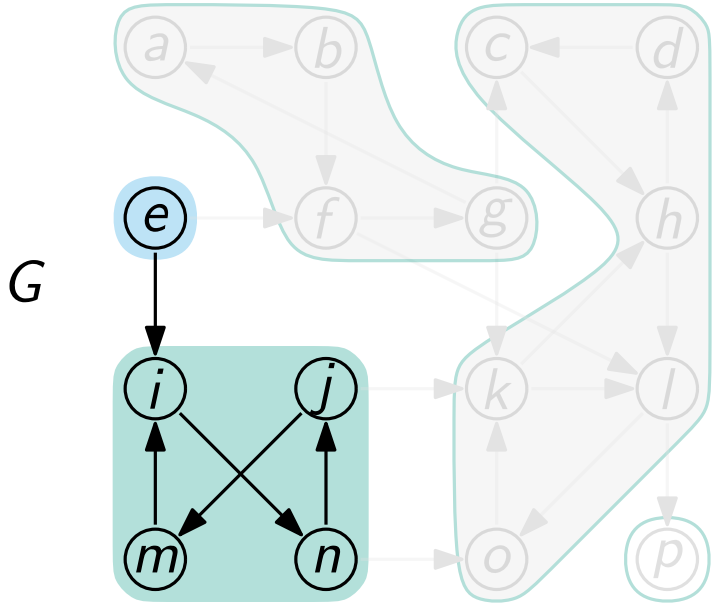
Beobachtung: Die SCCs eines Graphen bilden einen azyklischen Graphen.

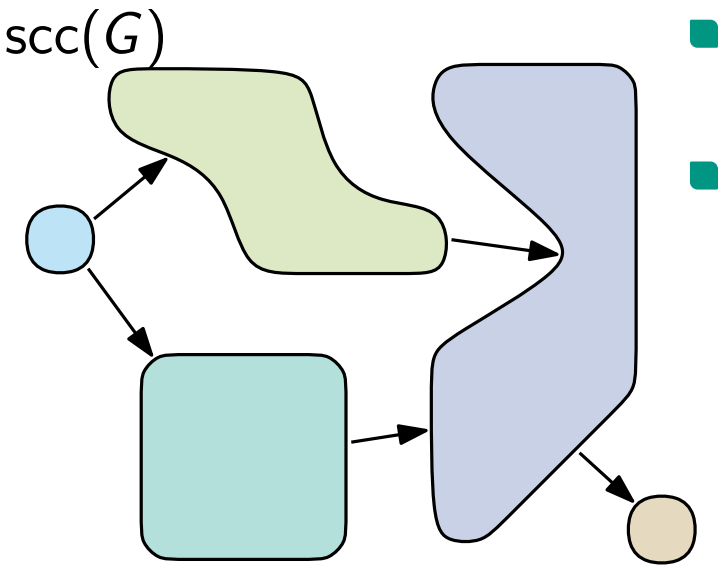


- Senken-Komponente: Senke in $\mathrm{scc}(G)$
- Algorithmus:
  - solange $V \neq \emptyset$: finde Knoten v in Senken-Komponente
    - finde SCC C von v, lösche C

Idee für Algorithmus – Beispiel

Beobachtung: Die SCCs eines Graphen bilden einen azyklischen Graphen.

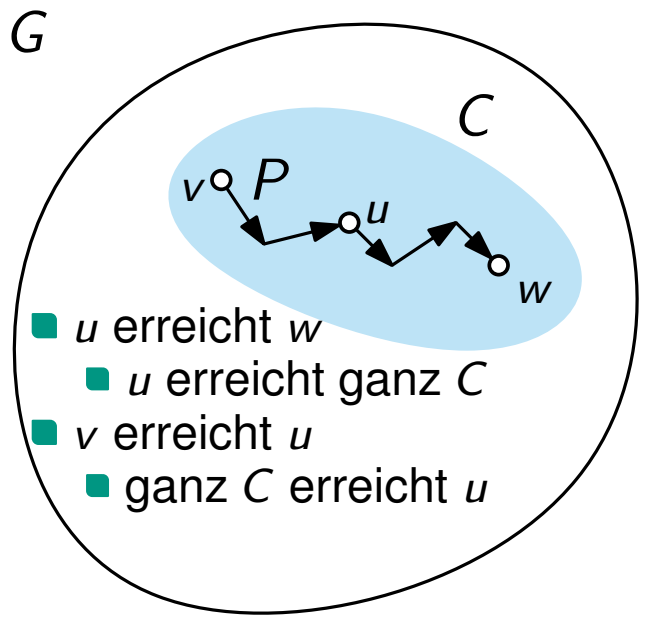




- Senken-Komponente: Senke in $\mathrm{scc}(G)$
- Algorithmus:
  - solange $V \neq \emptyset$: finde Knoten v in Senken-Komponente
    - finde SCC C von v, lösche C

SCCs – Struktur

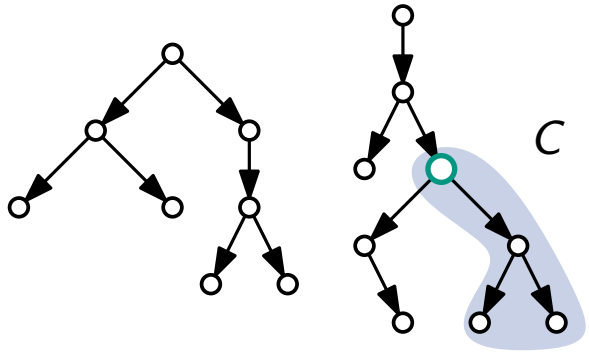
Beobachtung: Seien v, w Knoten in der gleichen SCC C und P ein Pfad von v nach w. Dann liegt P auch in C.



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 v mit $\text{parent}(v) \notin C$ oder $\text{parent}(v) = \perp$.



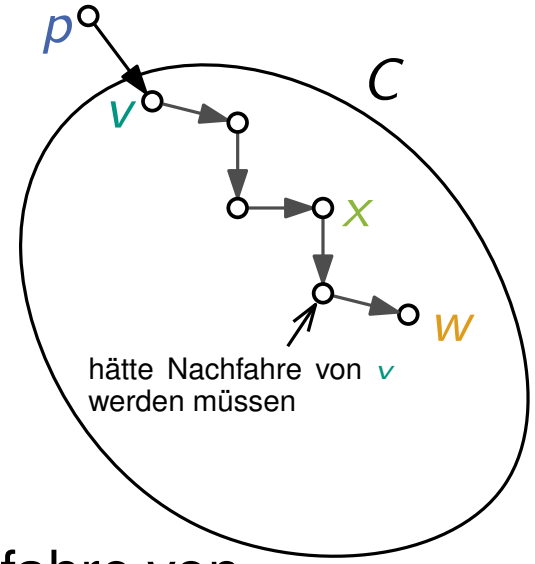
SCCs – Struktur

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 v mit $\text{parent}(v) \notin C$ oder $\text{parent}(v) = \perp$.

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

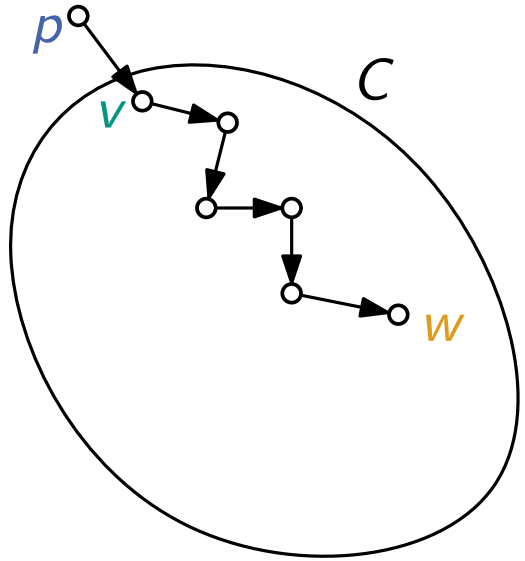
Beobachtung: Seien v, w Knoten in der gleichen SCC C und P ein Pfad von v nach w. Dann liegt P auch in C.

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

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

□



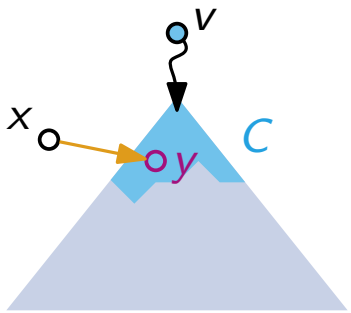
SCCs – Knoten in Senken Komponente finden

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 es gibt $(x, y) \in E$ 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 ↯

□



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

SCCs – Knoten in Senken Komponente finden 2

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

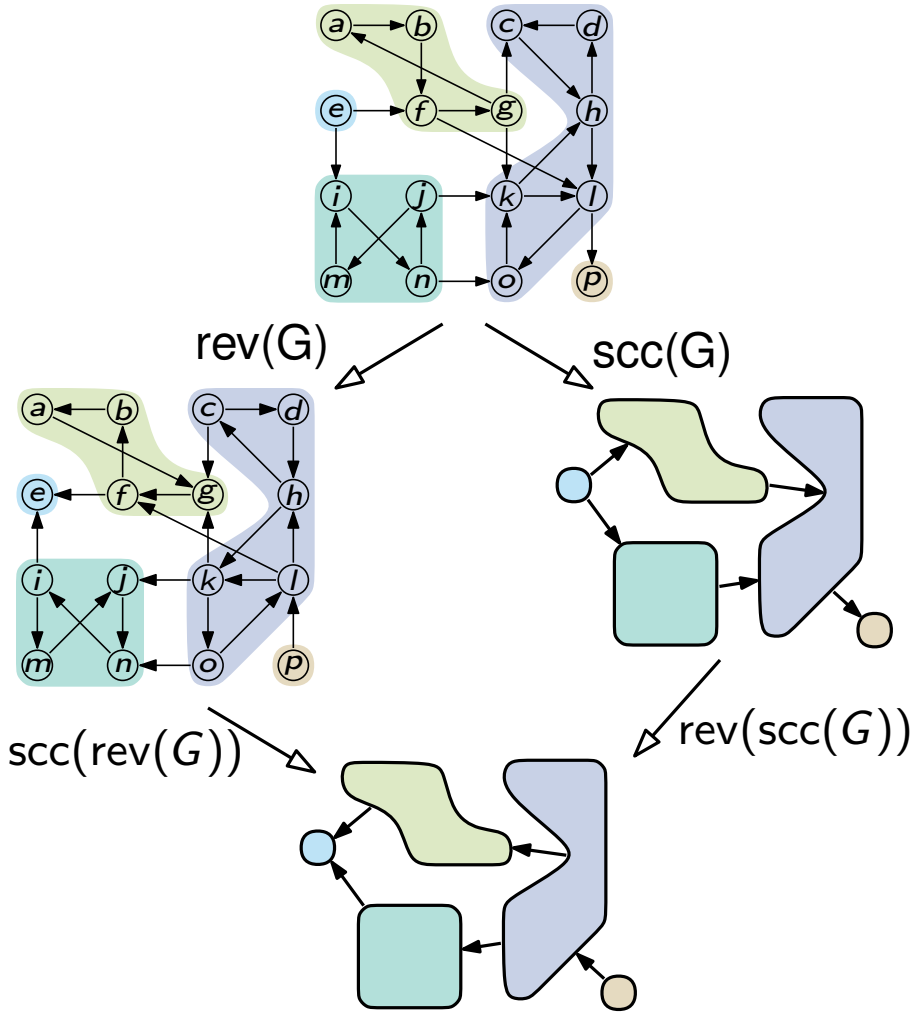
Beobachtung: $\text{rev}(\text{scc}(G)) = \text{scc}(\text{rev}(G))$

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

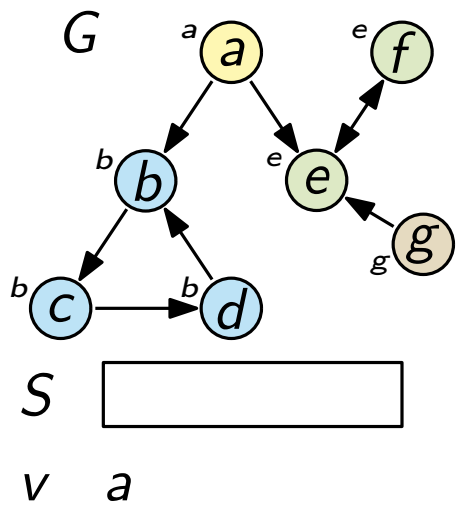
Idee:

- 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.scc := ⊥
    // 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 := pop(S)
        if v.scc = ⊥ then
            sccDFS(G, v, v)
```




```
revDFS(Graph G, Stack S, Node v)
    mark v
    for Node u with v ∈ N(u) do
        if u is unmarked then
            revDFS(G, S, u)
    S.push(v)
```

```
sccDFS(Graph G, Node v, Node r)
    v.scc = r
    for Node u in N(v) do
        if u.scc = ⊥ then
            sccDFS(G, u, r)
```

Theorem: Es gibt einen Algorithmus, der die SCCs eines gerichteten Graphen in $\mathcal{O}(n+m)$ Zeit berechnet.

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 $m+1 \in M$:

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

$2(m+1) - 1 + m^2 = m^2 + 2m + 1 = (m+1)^2$

Folgerung $M = \mathbb{N}$

Wieso ist das formal korrekt?

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

- $1 \in M$; und
- für alle $m \in M$ ist auch $m+1 \in M$,

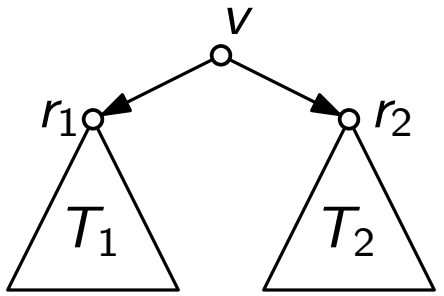
dann gilt $M = \mathbb{N}$.

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

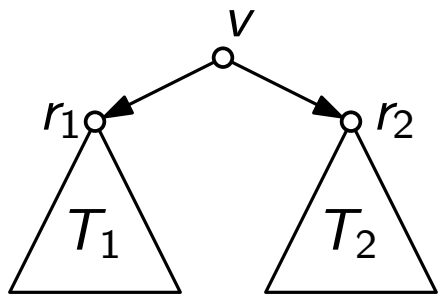
Sei M eine Menge von Binärbäumen. Falls gilt

- 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

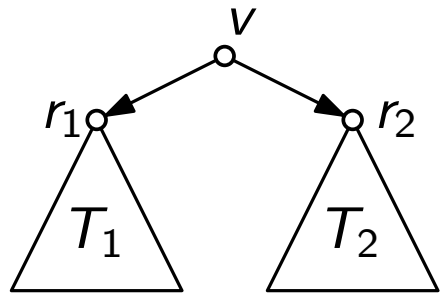
Satz über Balance
Sei T ein Binärbaum mit n Knoten, sodass für jeden Teilbaum T' mit Kindbäumen L und R gilt $|h(L) - h(R)| \leq 1$.
Dann gilt $\sqrt{2}^{h(T)-1} \leq n$.

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. struktureller 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

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}
```

Topologische Sortierung

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

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

```
if |V| = 1 then
    return ⟨V[0]⟩
source := findSource(G)
G' := removeNode(G, source)
return ⟨source⟩ + ⟨topoSort(G')⟩
```