Script generated by TTT

Title: Täubig: GAD (10.07.2012)

Date: Tue Jul 10 14:31:28 CEST 2012

Duration: 87:02 min

Pages: 35

Knoten-Zusammenhang

Definition

Ein ungerichteter Graph $G = (V, E)$ heißt k-fach zusammenhängend (oder genauer gesagt k-*knoten*zusammenhängend), falls

- $|V| > k$ und
- für jede echte Knotenteilmenge $X \subset V$ mit $|X| < k$ der Graph $G - X$ zusammenhängend ist.

Bemerkung:

- "zusammenhängend" ist im wesentlichen gleichbedeutend mit "1-knotenzusammenhängend"

  Ausnahme: Graph mit nur einem Knoten ist zusammenhängend, aber nicht 1-zusammenhängend

Artikulationsknoten und Blöcke

Definition

Ein Knoten v eines Graphen G heißt Artikulationsknoten (engl. *cut-vertex*), wenn sich die Anzahl der Zusammenhangskomponenten von G durch das Entfernen von v erhöht.

Definition

Die Zweifachzusammenhangskomponenten eines Graphen sind die maximalen Teilgraphen, die 2-fach zusammenhängend sind.

Ein Block ist ein maximaler zusammenhängender Teilgraph, der keinen Artikulationsknoten enthält.

D.h. die Menge der Blöcke besteht aus den Zweifachzusammenhangskomponenten, den Brücken (engl. *cut edges*), sowie den isolierten Knoten.

Artikulationsknoten und Blöcke

Definition

Ein Knoten v eines Graphen G heißt Artikulationsknoten (engl. *cut-vertex*), wenn sich die Anzahl der Zusammenhangskomponenten von G durch das Entfernen von v erhöht.

Definition

Die Zweifachzusammenhangskomponenten eines Graphen sind die maximalen Teilgraphen, die 2-fach zusammenhängend sind.

Ein Block ist ein maximaler zusammenhängender Teilgraph, der keinen Artikulationsknoten enthält.

D.h. die Menge der Blöcke besteht aus den Zweifachzusammenhangskomponenten, den Brücken (engl. *cut edges*), sowie den isolierten Knoten.

Blöcke und DFS

Modifizierte DFS nach R. E. Tarjan:

- `num[v]`: DFS-Nummer von v
- `low[v]`: minimale Nummer `num[w]` eines Knotens w, der von v aus über beliebig viele (≥ 0) Baumkanten abwärts, evt. gefolgt von einer einzigen Rückwärtskante erreicht werden kann

- `low[v]`: Minimum von
  - `num[v]`
  - `low[w]`, wobei w ein Kind von v im DFS-Baum ist (Baumkante)
  - `num[w]`, wobei $\{v, w\}$ eine Rückwärtskante ist

Artikulationsknoten und DFS

Lemma

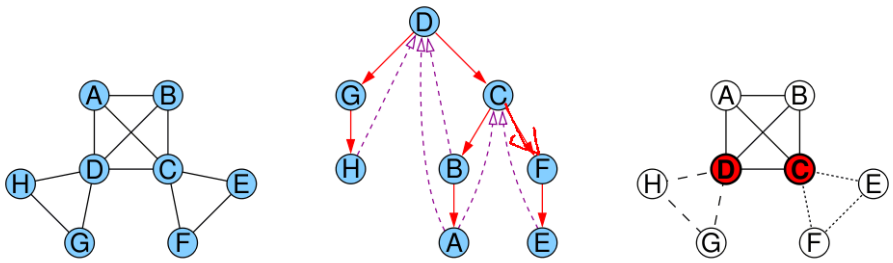
Sei $G = (V, E)$ ein ungerichteter, zusammenhängender Graph und T ein DFS-Baum in G.

Ein Knoten $a \in V$ ist genau dann ein Artikulationsknoten, wenn

- *a die Wurzel von T ist und mindestens 2 Kinder hat, oder*
- *a nicht die Wurzel von T ist und es ein Kind b von a mit `low[b]`$\geq$`num[a]` gibt.*

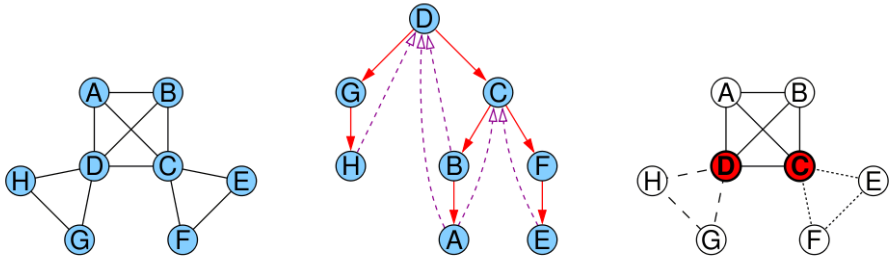
Artikulationsknoten und Blöcke per DFS

- bei Aufruf der DFS für Knoten v wird `num[v]` bestimmt und `low[v]` mit `num[v]` initialisiert
- nach Besuch eines Nachbarknotens w:
  Update von `low[v]` durch Vergleich mit
  - `low[w]` nach Rückkehr vom rekursiven Aufruf, falls (v, w) eine Baumkante war
  - `num[w]`, falls (v, w) eine Rückwärtskante war



Artikulationsknoten und Blöcke per DFS

- Kanten werden auf einem anfangs leeren Stack gesammelt
- Baumkanten kommen vor dem rekursiven Aufruf auf den Stack
- Rückwärtskanten werden direkt auf den Stack gelegt
- nach Rückkehr von einem rekursiven Aufruf werden im Fall `low[w]`$\geq$`num[v]` die obersten Kanten vom Stack bis einschließlich der Baumkante $\{v, w\}$ entfernt und bilden den nächsten Block



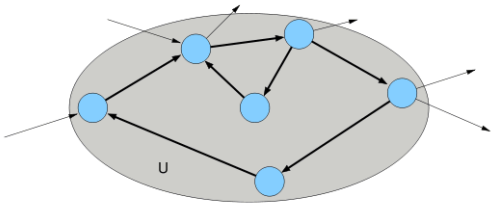
Starke Zusammenhangskomponenten

Definition

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

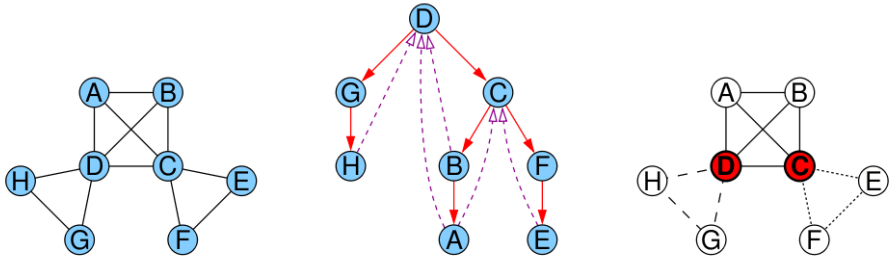
Knotenteilmenge $U \subseteq V$ heißt stark zusammenhängend genau dann, wenn für alle $u, v \in U$ ein gerichteter Pfad von u nach v in G existiert.

Für Knotenteilmenge $U \subseteq V$ heißt der induzierte Teilgraph $G[U]$ starke Zusammenhangskomponente von G, wenn U stark zusammenhängend und (inklusions-)maximal ist.



Artikulationsknoten und Blöcke per DFS

- Kanten werden auf einem anfangs leeren Stack gesammelt
- Baumkanten kommen vor dem rekursiven Aufruf auf den Stack
- Rückwärtskanten werden direkt auf den Stack gelegt
- nach Rückkehr von einem rekursiven Aufruf werden im Fall `low[w]≥num[v]` die obersten Kanten vom Stack bis einschließlich der Baumkante $\{v, w\}$ entfernt und bilden den nächsten Block



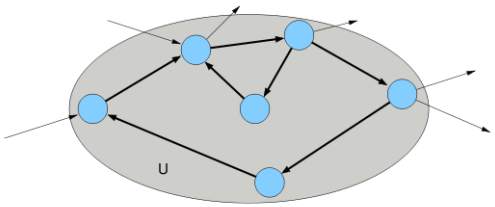
Starke Zusammenhangskomponenten

Definition

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

Knotenteilmenge $U \subseteq V$ heißt stark zusammenhängend genau dann, wenn für alle $u, v \in U$ ein gerichteter Pfad von u nach v in G existiert.

Für Knotenteilmenge $U \subseteq V$ heißt der induzierte Teilgraph $G[U]$ starke Zusammenhangskomponente von G, wenn U stark zusammenhängend und (inklusions-)maximal ist.



Starke Zusammenhangskomponenten

Beobachtungen:

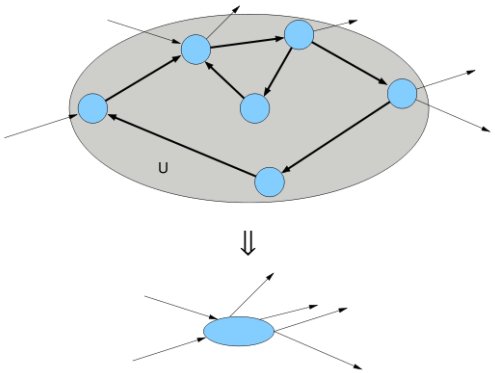
- Knoten $x, y \in V$ sind stark zusammenhängend, falls beide Knoten auf einem gemeinsamen gerichteten Kreis liegen (oder $x = y$).
- Die starken Zusammenhangskomponenten bilden eine Partition der Knotenmenge.

  (im Gegensatz zu 2-Zhk. bei ungerichteten Graphen, wo nur die Kantenmenge partitioniert wird, sich aber zwei verschiedene 2-Zhk. in einem Knoten überlappen können)

Starke Zusammenhangskomponenten

Beobachtungen:

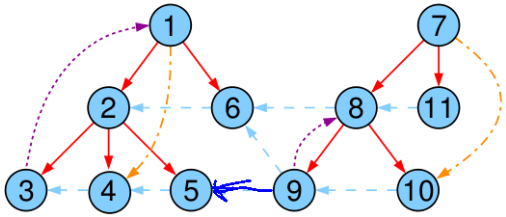
- Schrumpft man alle starken Zusammenhangskomponenten zu einzelnen (Super-)Knoten, ergibt sich ein DAG.



Starke Zhk. und DFS / Variante 1

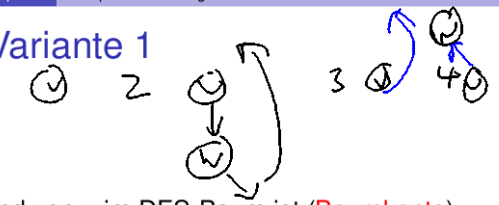
Modifizierte DFS nach R. E. Tarjan

- `num[v]`: DFS-Nummer von v
- In einer starken Zhk. heißt der Knoten mit kleinster DFS-Nummer Wurzel der starken Zhk.
- `low[v]`: minimales `num[w]` eines Knotens w, der von v aus über beliebig viele (≥ 0) Baumkanten abwärts, evt. gefolgt von einer einzigen Rückwärtskante oder einer Querkante zu einer ZHK, deren Wurzel echter Vorfahre von v ist, erreicht werden kann

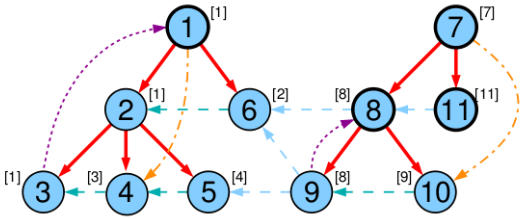


Starke Zhk. und DFS / Variante 1

Modifizierte DFS nach R. E. Tarjan



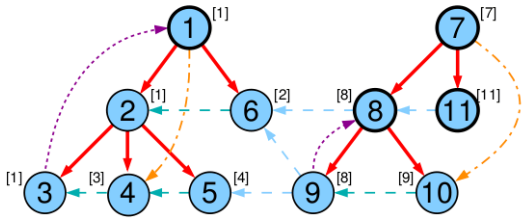
- `low[v]`: Minimum von
  - `num[v]`
  - `low[w]`, wobei w ein Kind von v im DFS-Baum ist (Baumkante)
  - `num[w]`, wobei $\{v, w\}$ eine Rückwärtskante ist
  - `num[w]`, wobei $\{v, w\}$ eine Querkante ist und die Wurzel der starken Zusammenhangskomponente von w ist Vorfahre von v

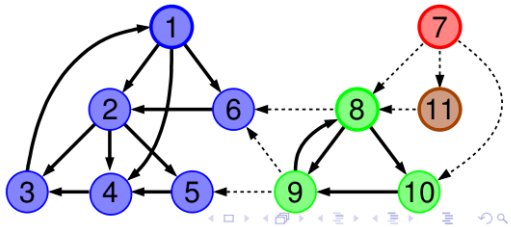


Starke Zhk. und DFS / Variante 1

Modifizierte DFS nach R. E. Tarjan

- Knoten v ist genau dann Wurzel einer starken Zusammenhangskomponente, wenn `num[v]`=`low[v]`





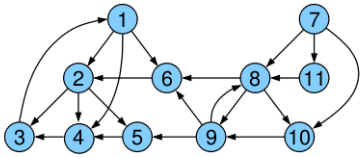
Starke Zhk. und DFS / Variante 2

Idee:

- beginne mit Graph ohne Kanten, jeder Knoten ist eigene SCC
- füge nach und nach einzelne Kanten ein

$\Rightarrow$ aktueller (current) Graph $G_c = (V, E_c)$

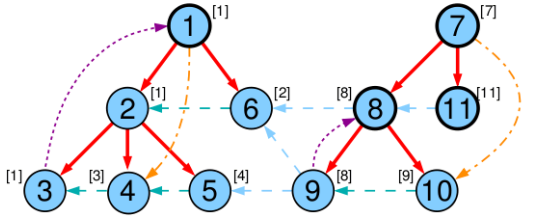
- Update der starken Zusammenhangskomponenten (SCCs)

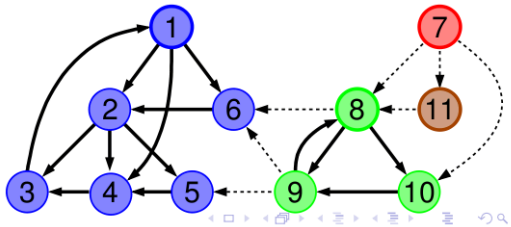


Starke Zhk. und DFS / Variante 1

Modifizierte DFS nach R. E. Tarjan

- Knoten v ist genau dann Wurzel einer starken Zusammenhangskomponente, wenn `num[v]=low[v]`





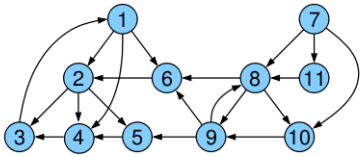
Starke Zhk. und DFS / Variante 2

Idee:

- beginne mit Graph ohne Kanten, jeder Knoten ist eigene SCC
- füge nach und nach einzelne Kanten ein

$\Rightarrow$ aktueller (current) Graph $G_c = (V, E_c)$

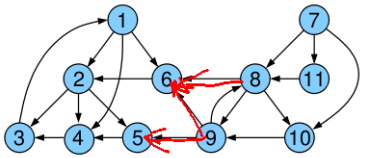
- Update der starken Zusammenhangskomponenten (SCCs)

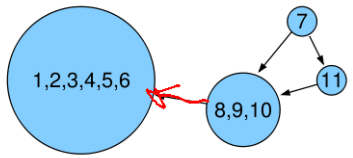


Starke Zhk. und DFS / Variante 2

Idee:

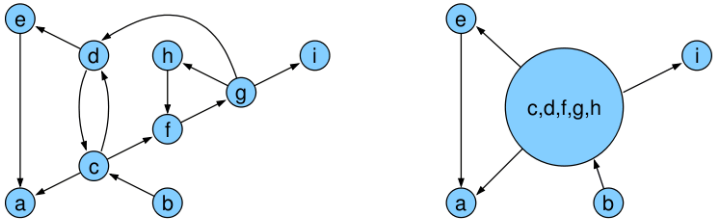
- betrachte geschrumpften (shrunken) Graph G_c^s: Knoten entsprechen SCCs von G_c, Kante (C, D) genau dann, wenn es Knoten $u \in C$ und $v \in D$ mit $(u, v) \in E_c$ gibt
- geschrumpfter Graph G_c^s ist ein DAG
- Ziel: Aktualisierung des geschrumpften Graphen beim Einfügen



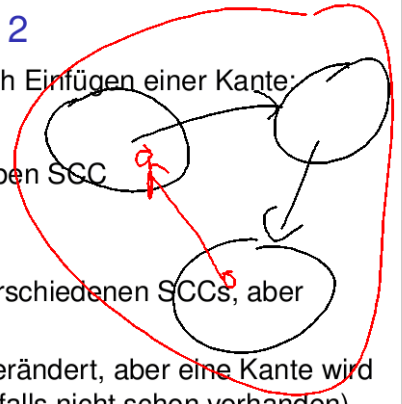


Starke Zhk. und DFS / Variante 2

Geschrumpfter Graph
(Beispiel aus Mehlhorn / Sanders)



Starke Zhk. und DFS / Variante 2



Update des geschrumpften Graphen nach Einfügen einer Kante:

3 Möglichkeiten:

- beide Endpunkte gehören zu derselben SCC

⇒ geschrumpfter Graph unverändert

- Kante verbindet Knoten aus zwei verschiedenen SCCs, aber schließt keinen Kreis

⇒ SCCs im geschrumpften Graph unverändert, aber eine Kante wird im geschrumpften Graph eingefügt (falls nicht schon vorhanden)

- Kante verbindet Knoten aus zwei verschiedenen SCCs und schließt einen oder mehrere Kreise

⇒ alle SCCs, die auf einem der Kreise liegen, werden zu einer einzigen SCC verschmolzen

Starke Zhk. und DFS / Variante 2

Prinzip:

- Tiefensuche
  - V_c schon markierte (entdeckte) Knoten
  - E_c schon gefundene Kanten
- 3 Arten von SCC: unentdeckt, offen, geschlossen
- unentdeckte Knoten haben Ein- / Ausgangsgrad Null in G_c

⇒ zunächst bildet jeder Knoten eine eigene unentdeckte SCC, andere SCCs enthalten nur markierte Knoten

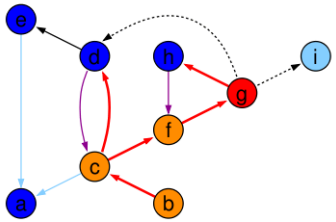
- SCCs mit mindestens einem aktiven Knoten (ohne finishNum) heißen offen
- SCC heißt geschlossen, falls sie nur fertige Knoten (mit finishNum) enthält
- Knoten in offenen / geschlossenen SCCs heißen offen / geschlossen

Starke Zhk. und DFS / Variante 2

- Knoten in geschlossenen SCCs sind immer fertig (mit finishNum)
- Knoten in offenen SCCs können fertig oder noch aktiv (ohne finishNum) sein
- Repräsentant einer SCC: Knoten mit kleinster dfsNum

Starke Zhk. und DFS / Variante 2

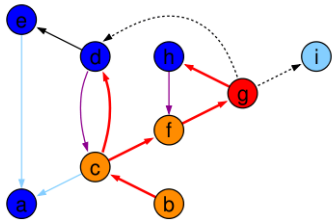
DFS-Snapshot:

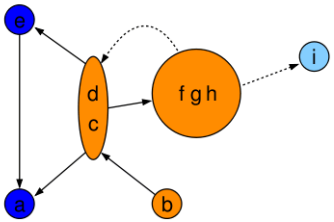


- erste DFS startete bei Knoten a, zweite bei b
- aktueller Knoten ist g, auf dem Stack liegen b, c, f, g
- (g, d) und (g, i) wurden noch nicht exploriert
- (d, c) und (h, f) sind Rückwärtskanten
- (c, a) und (e, a) sind Querkanten
- (b, c), (c, d), (c, f), (f, g) und (g, h) sind Baumkanten

Starke Zhk. und DFS / Variante 2

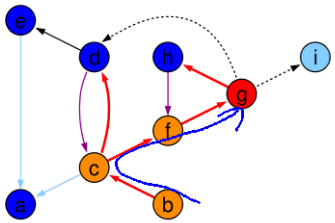
DFS-Snapshot mit geschrumpftem Graph:

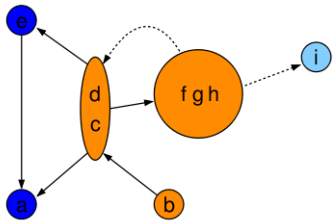




- unentdeckt: $\{i\}$ offen: $\{b\}$, $\{c, d\}$, $\{f, g, h\}$ geschlossen: $\{a\}$, $\{e\}$
- offene SCCs bilden Pfad im geschrumpften Graph
- aktueller Knoten gehört zur letzten SCC
- offene Knoten wurden in Reihenfolge b, c, d, f, g, h erreicht und werden von den Repräsentanten b, c und f genau in die offenen SCCs partitioniert

Starke Zhk. und DFS / Variante 2





Beobachtungen (Invarianten für G_c):

1. Pfade aus geschlossenen Knoten / SCCs führen immer zu geschlossenen Knoten / SCCs
2. Pfad zum aktuellen Knoten enthält die Repräsentanten aller offenen SCCs

   offene Komponenten bilden Pfad im geschrumpften Graph
3. Knoten der offenen SCCs in Reihenfolge der DFS-Nummern werden durch Repräsentanten in die offenen SCCs partitioniert

Starke Zhk. und DFS / Variante 2

Geschlossene SCCs von G_c sind auch SCCs in G:

- Sei v geschlossener Knoten und S / S_c seine SCC in G / G_c.
- zu zeigen: $S = S_c$
- G_c ist Subgraph von G, also $S_c \subseteq S$
- somit zu zeigen: $S \subseteq S_c$
- Sei w ein Knoten in S.

$\Rightarrow$ $\exists$ Kreis C durch v und w.

- Invariante 1: alle Knoten von C sind geschlossen und somit erledigt (alle ausgehenden Kanten exploriert)
- C ist in G_c enthalten, also $w \in S_c$
- damit gilt $S \subseteq S_c$, also $S = S_c$

Starke Zhk. und DFS / Variante 2

Vorgehen:

- Invarianten 2 und 3 helfen bei Verwaltung der offenen SCCs
- Knoten in offenen SCCs auf Stack oNodes
  (in Reihenfolge steigender dfsNum)
- Repräsentanten der offenen SCCs auf Stack oReps
- zu Beginn Invarianten gültig (alles leer)
- vor Markierung einer neuen Wurzel sind alle markierten Knoten erledigt, also keine offenen SCCs, beide Stacks leer
  dann: neue offene SCC für neue Wurzel s,
  s kommt auf beide Stacks

Starke Zhk. und DFS / Variante 2

Geschlossene SCCs von G_c sind auch SCCs in G:

- Sei v geschlossener Knoten und S / S_c seine SCC in G / G_c.
- zu zeigen: $S = S_c$
- G_c ist Subgraph von G, also $S_c \subseteq S$
- somit zu zeigen: $S \subseteq S_c$

- Sei w ein Knoten in S.

$\Rightarrow$ $\exists$ Kreis C durch v und w.

- Invariante 1: alle Knoten von C sind geschlossen und somit erledigt (alle ausgehenden Kanten exploriert)
- C ist in G_c enthalten, also $w \in S_c$
- damit gilt $S \subseteq S_c$, also $S = S_c$

Starke Zhk. und DFS / Variante 2

Geschlossene SCCs von G_c sind auch SCCs in G:

- Sei v geschlossener Knoten und S / S_c seine SCC in G / G_c.
- zu zeigen: $S = S_c$
- G_c ist Subgraph von G, also $S_c \subseteq S$
- somit zu zeigen: $S \subseteq S_c$

- Sei w ein Knoten in S.

$\Rightarrow$ $\exists$ Kreis C durch v und w.

- Invariante 1: alle Knoten von C sind geschlossen und somit erledigt (alle ausgehenden Kanten exploriert)
- C ist in G_c enthalten, also $w \in S_c$
- damit gilt $S \subseteq S_c$, also $S = S_c$

Starke Zhk. und DFS / Variante 2

Prinzip: betrachte Kante $e = (v, w)$

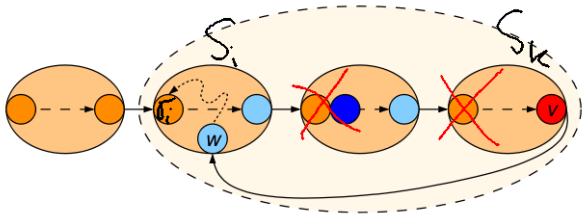
- Kante zu unbekanntem Knoten w (Baumkante):
  neue eigene offene SCC für w (w kommt auf oNodes und oReps)
- Kante zu Knoten w in geschlossener SCC (Nicht-Baumkante):
  von w gibt es keinen Weg zu v, sonst wäre die SCC von w noch nicht geschlossen (geschlossene SCCs sind bereits komplett), also SCCs unverändert
- Kante zu Knoten w in offener SCC (Nicht-Baumkante):
  falls v und w in unterschiedlichen SCCs liegen, müssen diese mit allen SCCs dazwischen zu einer einzigen SCC verschmolzen werden (durch Löschen der Repräsentanten)

Wenn Knoten keine ausgehenden Kanten mehr hat:

- Knoten fertig
- wenn Knoten Repräsentant seiner SCC ist, dann SCC schließen

Starke Zhk. und DFS / Variante 2

Vereinigung offener SCCs im Kreisfall:



- offene SCC entsprechen Ovalen, Knoten sortiert nach dfsNum
- alle Repräsentanten offener SCCs liegen auf Baumpfad zum aktuellen Knoten v in SCC S_k
- Nicht-Baumkante (v, w) endet an Knoten w in offener SCC S_i mit Repräsentant r_i
- Pfad von w nach r_i muss existieren (innerhalb SCC S_i)

$\Rightarrow$ Kante (v, w) vereinigt $S_i, \ldots, S_k$

Starke Zhk. und DFS / Variante 2

- init() {
  component = new int[n];
  oReps = ⟨ ⟩;
  oNodes = ⟨ ⟩;
  dfsCount = 1;
  }

- root(Node w) / traverseTreeEdge(Node v, Node w) {
  oReps.push(w); // Repräsentant einer neuen ZHK
  oNodes.push(w); // neuer offener Knoten
  dfsNum[w] = dfsCount;
  dfsCount++;
  }

Starke Zhk. und DFS / Variante 2

- traverseNonTreeEdge(Node v, Node w) {
  if (w ∈ oNodes) // verschmelze SCCs
  while (dfsNum[w] < dfsNum[oReps.top()])
  oReps.pop();
  }

- backtrack(Node u, Node v) {
  if (v == oReps.top()) { // v Repräsentant?
  oReps.pop(); // ja: entferne v
  do { // und offene Knoten bis v
  w = oNodes.pop();
  component[w] = v;
  } while (w!=v);
  }
  }

Starke Zhk. und DFS / Variante 2

Zeit: $O(n+m)$

Begründung:

- init, root: $O(1)$
- traverseTreeEdge: $(n-1) \times O(1)$
- backtrack, traverseNonTreeEdge:
  da jeder Knoten höchstens einmal in oReps und oNodes landet,
  insgesamt $O(n+m)$
- DFS-Gerüst: $O(n+m)$
- gesamt: $O(n+m)$