Algorithmen und Datenstrukturen

Prof. Dr. Ralf Möller

Universität zu Lübeck

Institut für Informationssysteme

Felix Kuhr (Übungen)

sowie viele Tutoren

Danksagung

Die nachfolgenden Präsentationen wurden mit ausdrücklicher Erlaubnis des Autors übernommen aus:

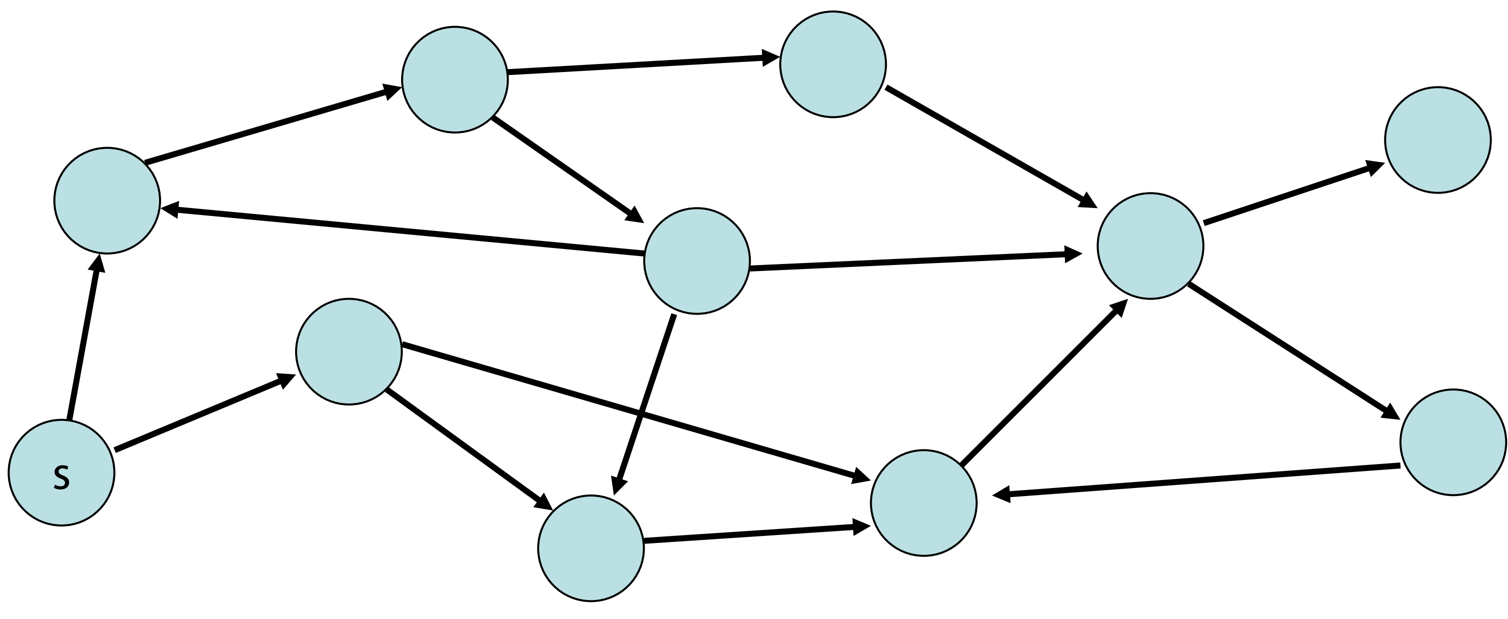
- „Effiziente Algorithmen und Datenstrukturen" (Kapitel 7,8,9) gehalten von Christian Scheideler an der TUM
  http://www14.in.tum.de/lehre/2008WS/ea/index.html.de

Es wurden umfangreiche Veränderungen vorgenommen

Fehler sind selbstverständlich uns zuzuschreiben

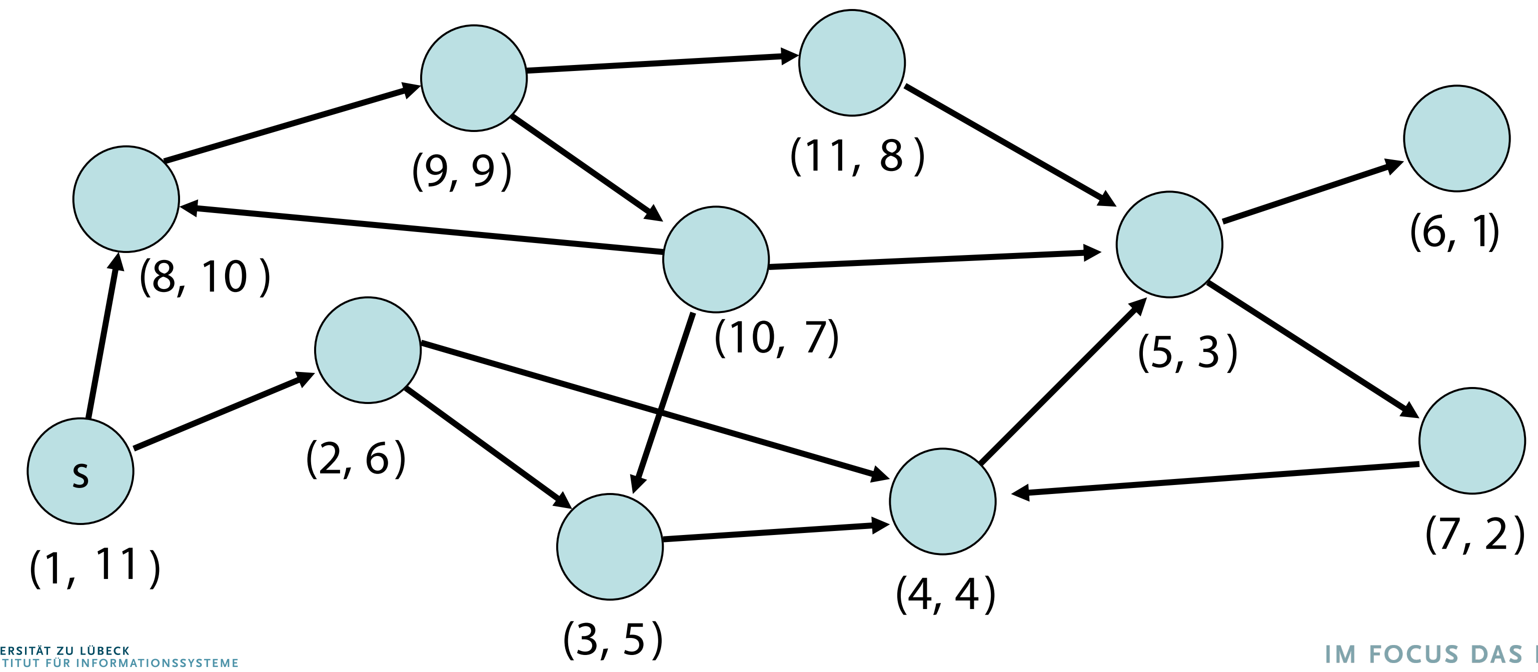
Tiefensuche

- Starte von einem Knoten s
- Exploriere Graph in die Tiefe
  (: aktuell, : noch aktiv, : fertig)



DFS-Nummerierung

- Exploriere Graph in die Tiefe
  (●: aktuell, ●: noch aktiv, ●: fertig)
- Paare (i,j): i: dfsNum, j: finishTime



Tiefensuche – Design Pattern

Übergeordnete Prozedur:

```
unmark all nodes
init()
for s∈V do  // stelle sicher, dass alle Knoten besucht werden
   if s is not marked then
      mark s
      root(s)
      DFS(s,s)    // s: Startknoten
```

```
Procedure DFS(u,v: Node)  // u: Vater von v
   for (v,w)∈E do
      if w is marked then handleNonTreeEdge(v,w)
      else traverseTreeEdge(v,w)
         mark w
         DFS(v,w)
   backtrack(u,v)
```

Könnte man das auch iterativ lösen?

Prozeduren in rot: noch zu spezifizieren

DFS-Nummerierung

Variablen:

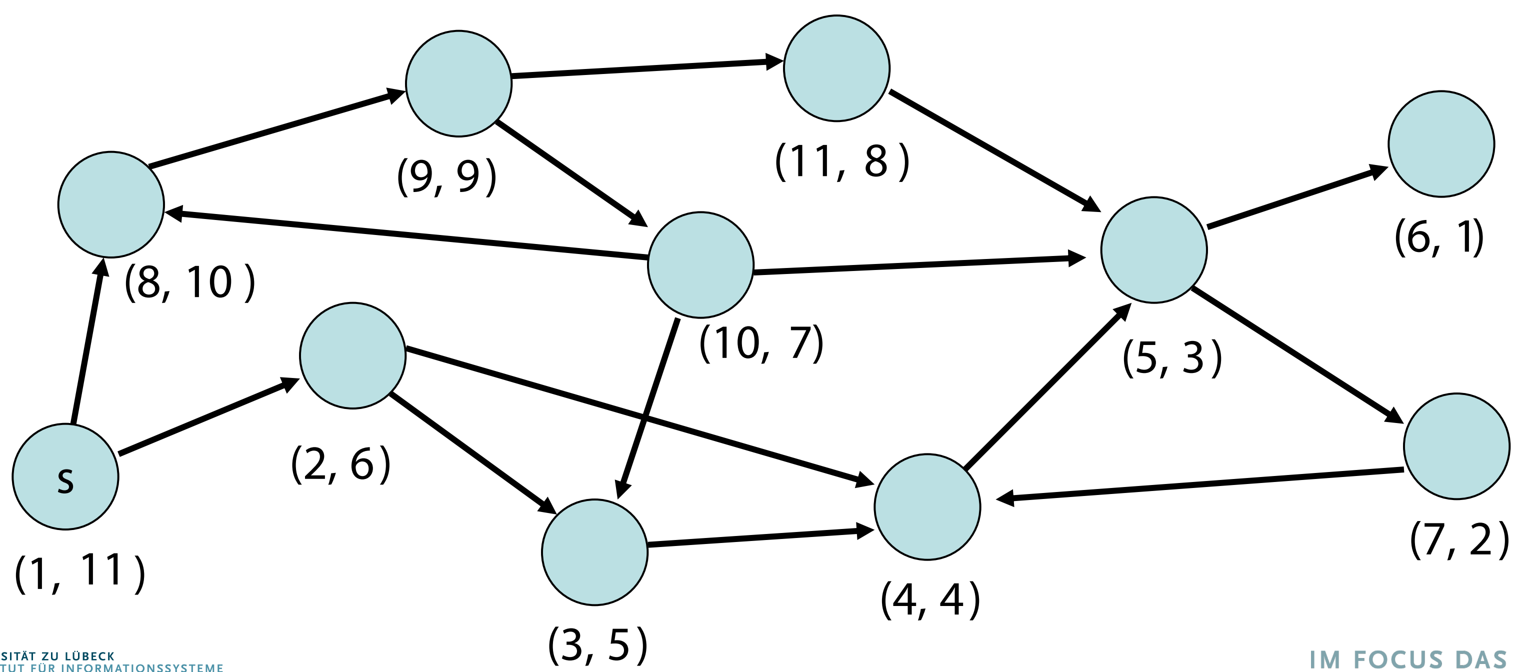
- dfsNum: Array [1..n] of ℕ // Zeitpunkt wenn Knoten
- finishTime: Array [1..n] of ℕ // Zeitpunkt wenn Knoten →
- dfsPos, finishingTime: ℕ // Zähler

Code-Stücke:

- init():
  dfsPos:=1; finishingTime:=1
- root(s):
  dfsNum[s]:=dfsPos; dfsPos:=dfsPos+1
- traverseTreeEdge(v,w):
  dfsNum[w]:=dfsPos; dfsPos:=dfsPos+1
- handleNonTreeEdge(v,w):
  -
- backtrack(u,v):
  finishTime[v]:=finishingTime; finishingTime := finishingTime+1

DFS-Nummerierung

- Exploriere Graph in die Tiefe
  (🟠: aktuell, 🔴: noch aktiv, 🔵: fertig)
- Paare (i,j): i: dfsNum, j: finishTime
- DFS-Nummerierung in O(n+m)



DFS-Nummerierung

Ordnung < auf den Knoten:

u<v gdw. dfsNum[u]<dfsNum[v]

Lemma 1: Die Knoten im DFS-Rekursionsstack (alle ● Knoten) sind sortiert bezüglich <.

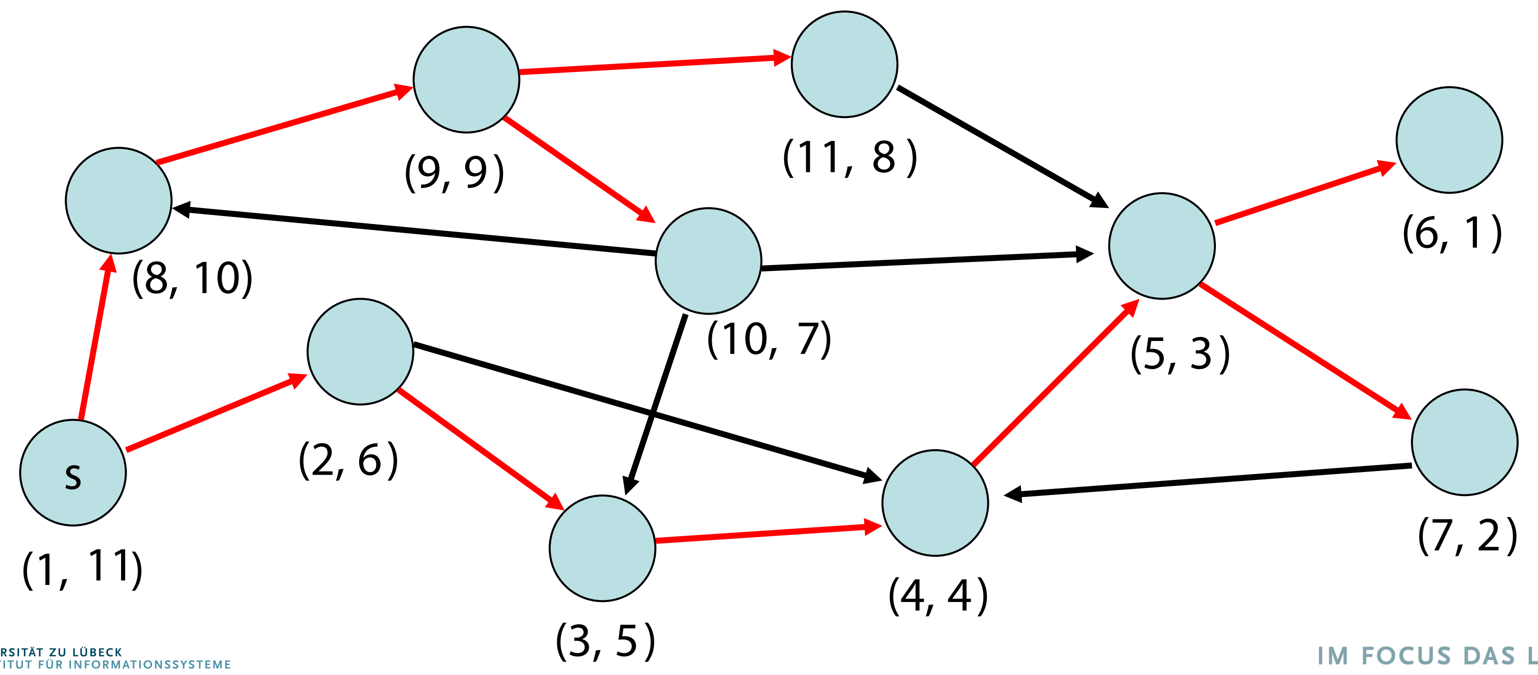
Beweis:

dfsPos wird nach jeder Zuweisung von dfsNum erhöht. Jeder neue aktive Knoten hat also immer die höchste dfsNum.

DFS-Nummerierung

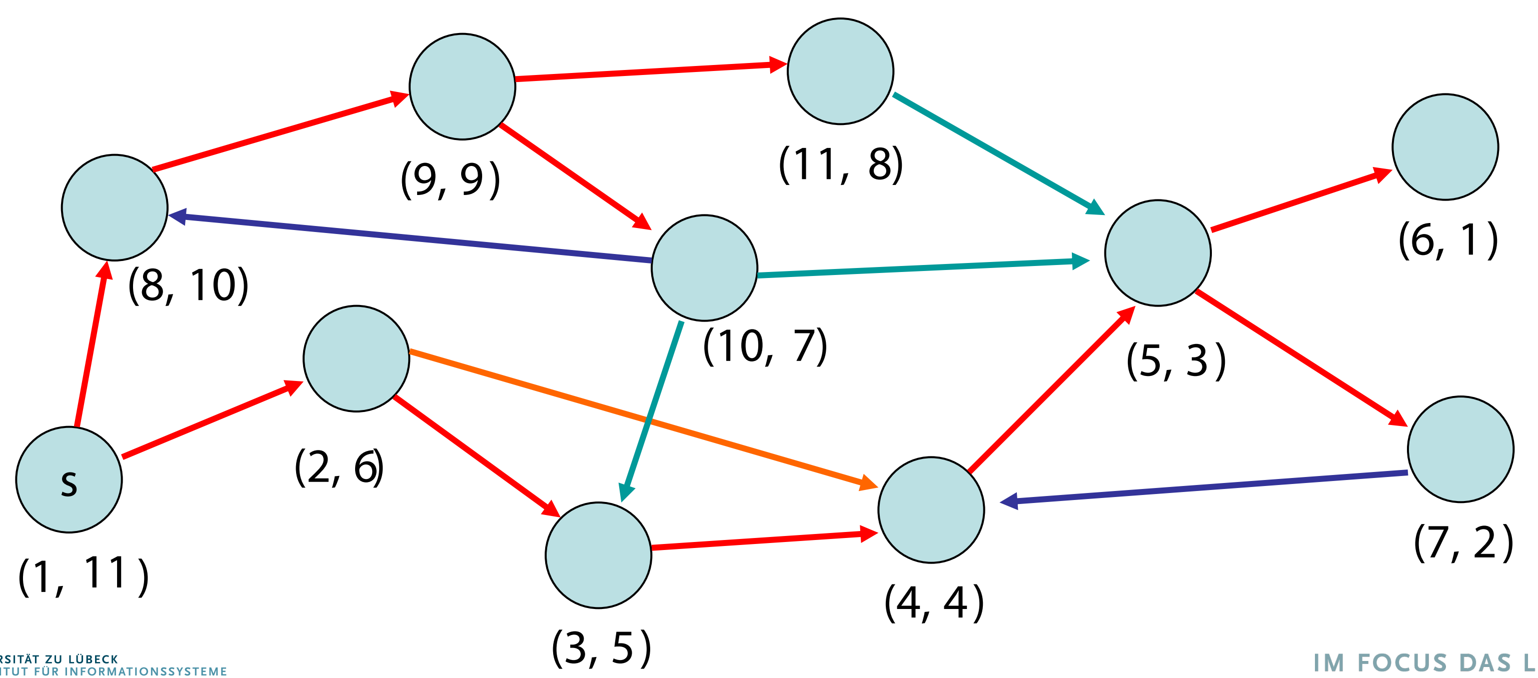
Überprüfung von Lemma 1:

- Rekursionsstack: roter Pfad von s
- Paare (i,j): i: dfsNum, j: finishTime



DFS-Nummerierung

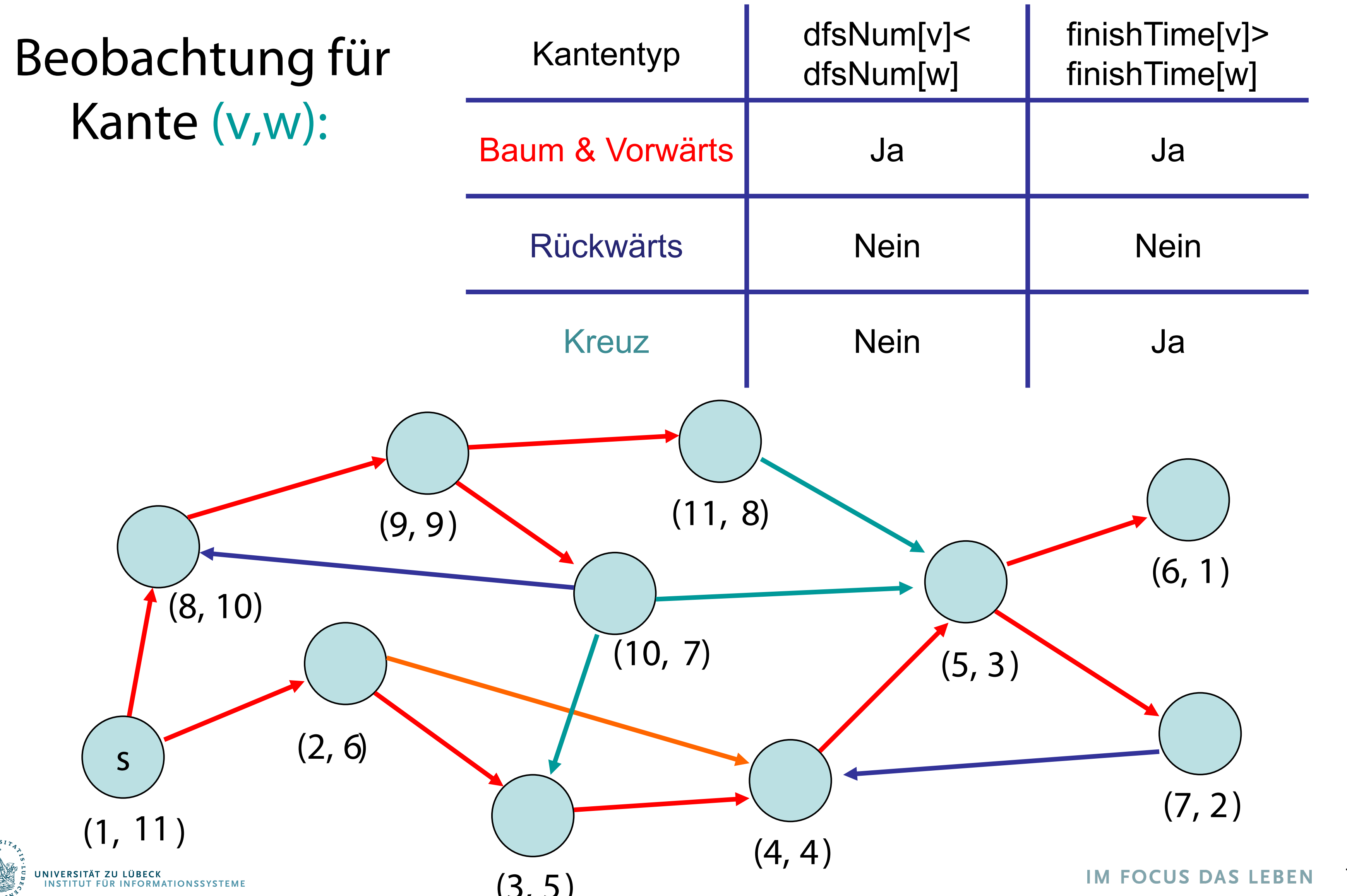
- Baumkante: zum Kind
- Vorwärtskante: zu einem Nachkommen
- Rückwärtskante: zu einem Vorfahr (siehe handleNonTreeEdge)
- Kreuzkante: alle sonstige Kanten (siehe handleNonTreeEdge)



DFS-Nummerierung

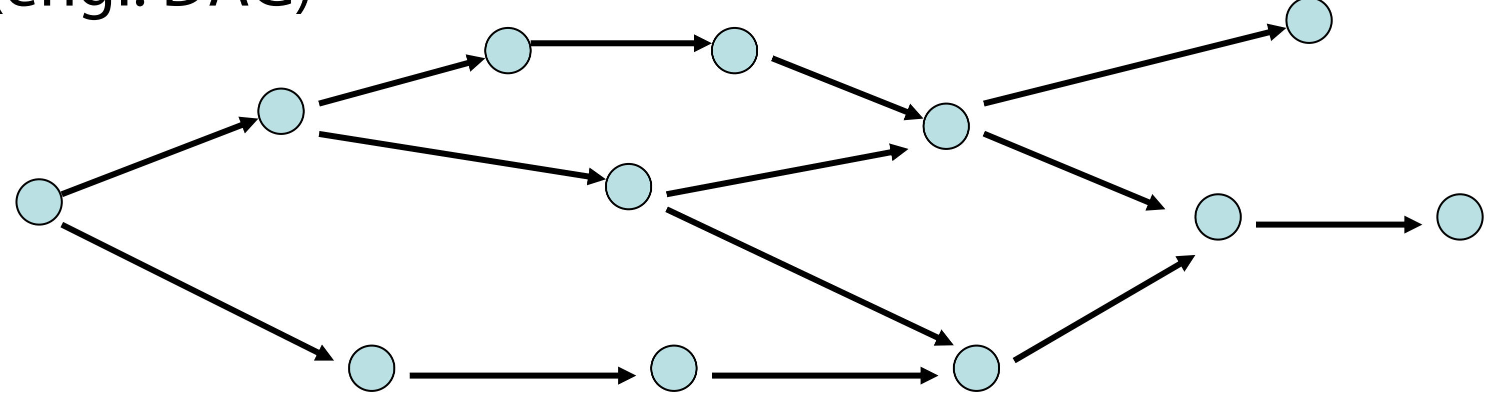
Beobachtung für Kante (v,w):

Kantentyp	dfsNum[v]< dfsNum[w]	finishTime[v]> finishTime[w]
Baum & Vorwärts	Ja	Ja
Rückwärts	Nein	Nein
Kreuz	Nein	Ja



DAGs

- Erkennung eines azyklischen gerichteten Graphen (engl. DAG)



Merkmal: keine gerichteten Kreise

Anwendung der DFS-Nummerierung:

In einem DAG gibt es keine Rückwärtskanten

DFS-Nummerierung

Behauptung: Folgende Aussagen sind äquivalent:

1. G ist ein DAG
2. DFS enthält keine Rückwärtskante
3. $\forall (v,w) \in E$: finishTime[v]>finishTime[w]

Beweis (2. ⇔ 3.):

2. ⇔ 3.: folgt aus Tabelle

Kantentyp	dfsNum[v]< dfsNum[w]	finishTime[v]> finishTime[w]
Baum & Vorwärts	Ja	Ja
Rückwärts	Nein	Nein
Kreuz	Nein	Ja

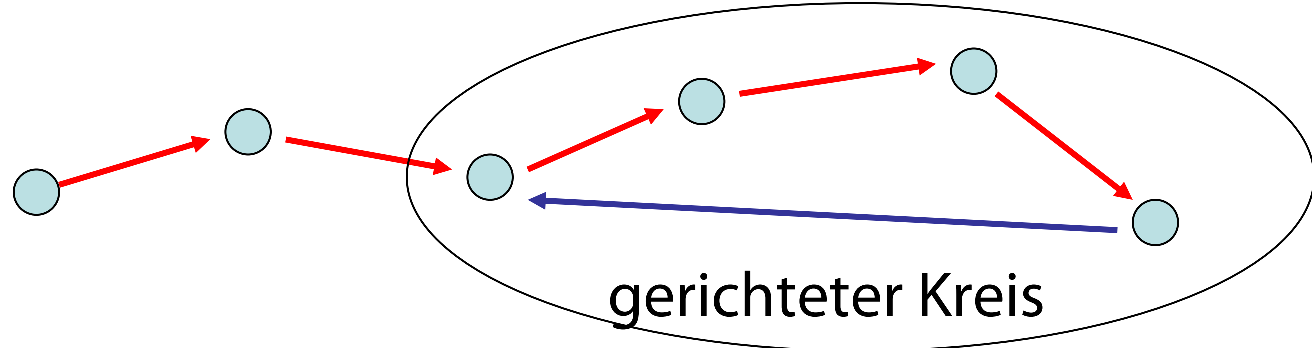
DFS-Nummerierung

Behauptung: Folgende Aussagen sind äquivalent :

1. G ist ein DAG
2. DFS enthält keine Rückwärtskante
3. $\forall (v,w) \in E$: finishTime[v]>finishTime[w]

Beweis (1. ⇒ 2.):

kontrapositiv ¬ 2. ⇒ ¬ 1.



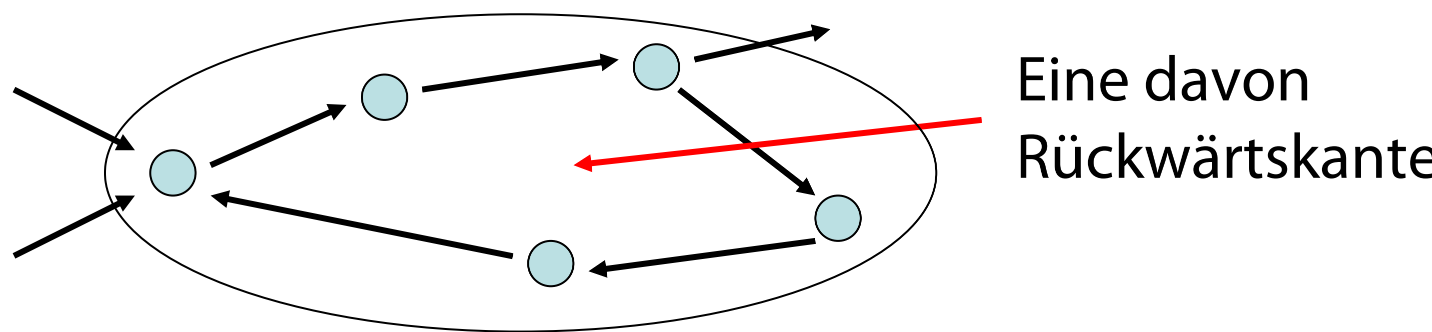
DFS-Nummerierung

Behauptung: Folgende Aussagen sind äquivalent :

1. G ist ein DAG
2. DFS enthält keine Rückwärtskante
3. $\forall (v,w) \in E : \text{finishTime}[v] > \text{finishTime}[w]$

Beweis (2. ⇒ 1.):

kontrapositiv ¬ 1. ⇒ ¬ 2.



DFS-Nummerierung

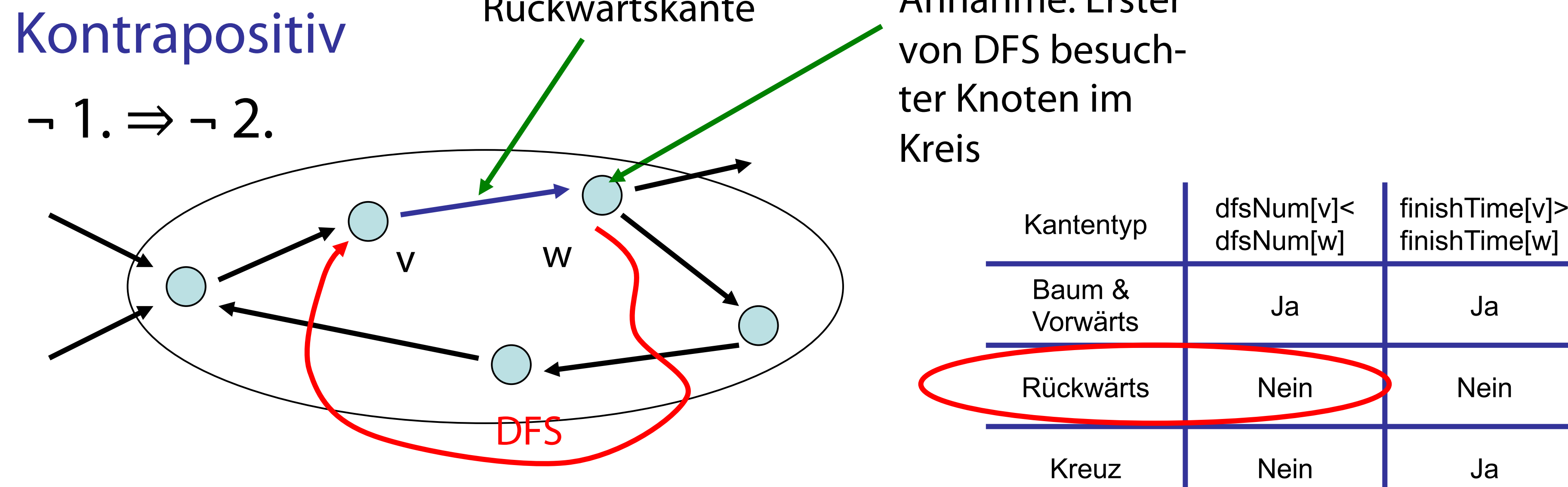
Behauptung: Folgende Aussagen sind äquivalent :

1. G ist ein DAG
2. DFS enthält keine Rückwärtskante
3. $\forall (v,w) \in E : \text{finishTime}[v] > \text{finishTime}[w]$

Beweis (2. ⇒ 1.):

Kontrapositiv

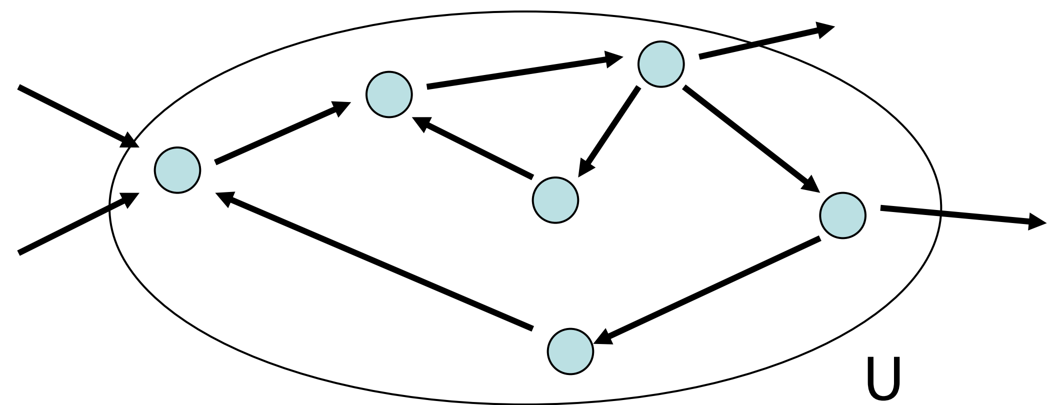
$\neg 1. \Rightarrow \neg 2.$



Kantentyp	dfsNum[v]< dfsNum[w]	finishTime[v]> finishTime[w]
Baum & Vorwärts	Ja	Ja
Rückwärts	Nein	Nein
Kreuz	Nein	Ja

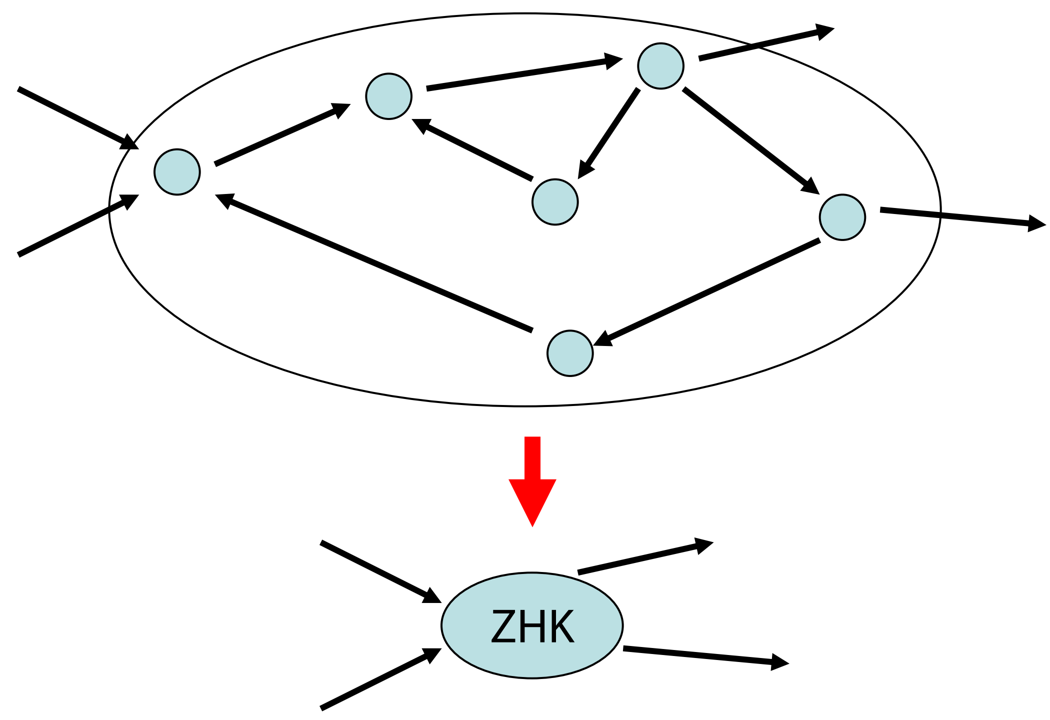
Starke ZHKs

Definition: Sei $G=(V,E)$ ein gerichteter Graph.
$U \subseteq V$ ist eine **starke Zusammenhangskomponente** (ZHK) von V gdw. für alle $u,v \in U$ gibt es einen gerichteten Weg von u nach v in G und U maximal



Starke ZHKs

Beobachtung: Schrumpft man starke ZHKs zu einzelnen Knoten, dann ergibt sich DAG.



Starke ZHKs - Beispiel

Starke ZHKs - Beispiel

DAG

Starke ZHKs

Ziel: Finde alle starken ZHKs im Graphen in $O(n+m)$ Zeit
(n: #Knoten, m: #Kanten)

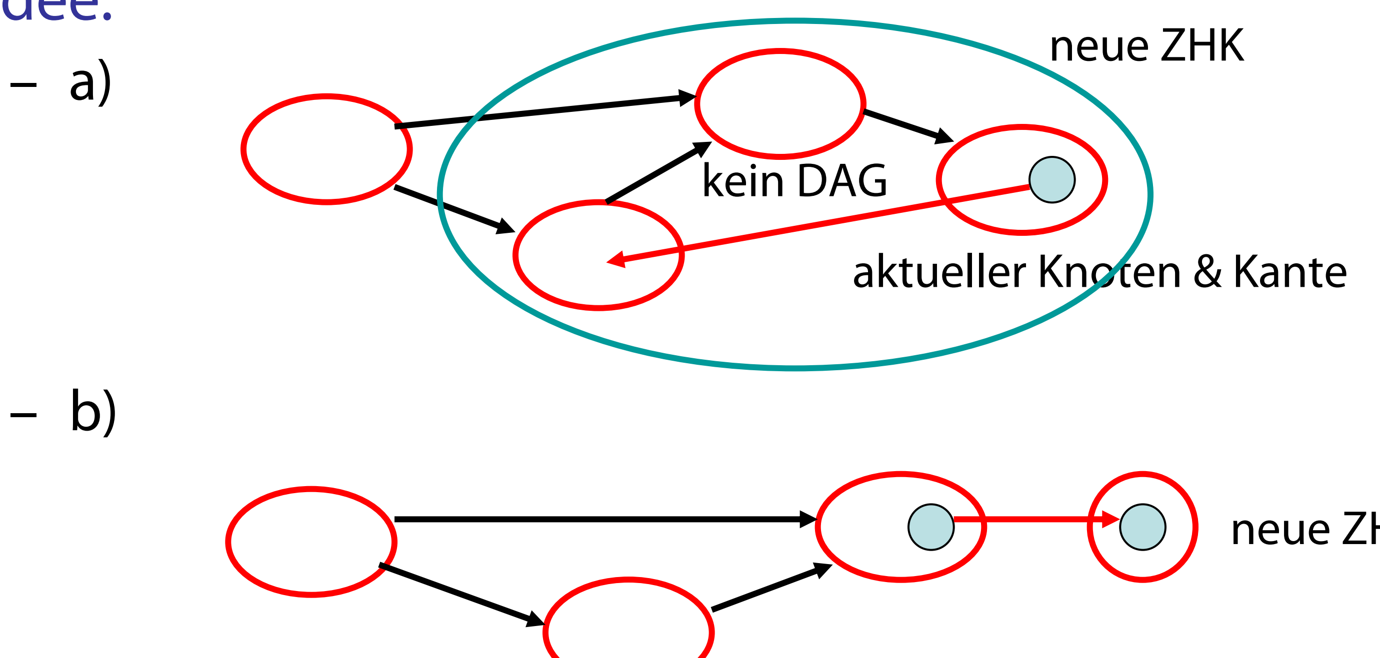
Strategie: Verwende DFS-Verfahren mit
component: Array [1..n] of 1..n

Am Ende: component[v]=component[w] ⇔
v und w sind in derselben starken ZHK

Robert Tarjan: *Depth-first search and linear graph algorithms*.
In: *SIAM Journal on Computing*. Bd. 1, Nr. 2, S. 146-160, **1972**

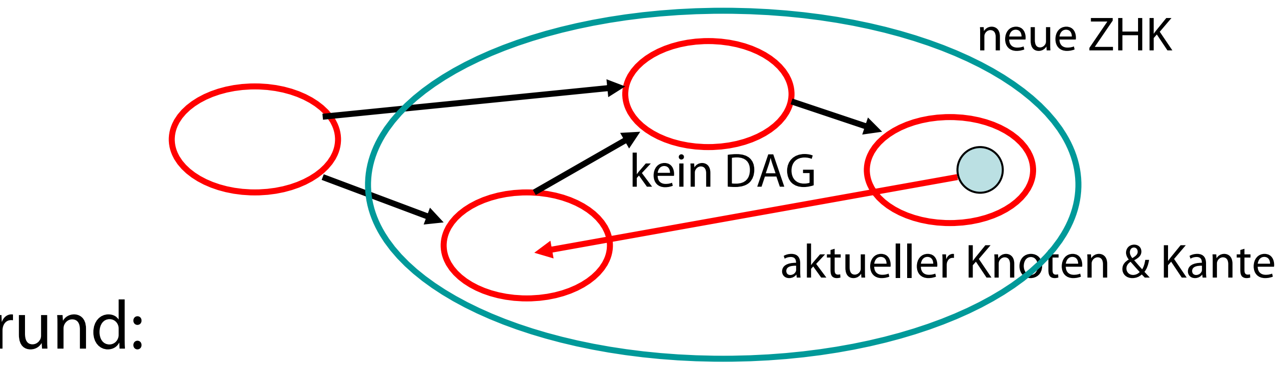
Starke ZHKs

- Betrachte DFS auf $G=(V,E)$
- Sei $G_c=(V_c,E_c)$ bereits besuchter Teilgraph von G
- Ziel: bewahre starke ZHKs in G_c
- Idee:
  - a)
  - b)

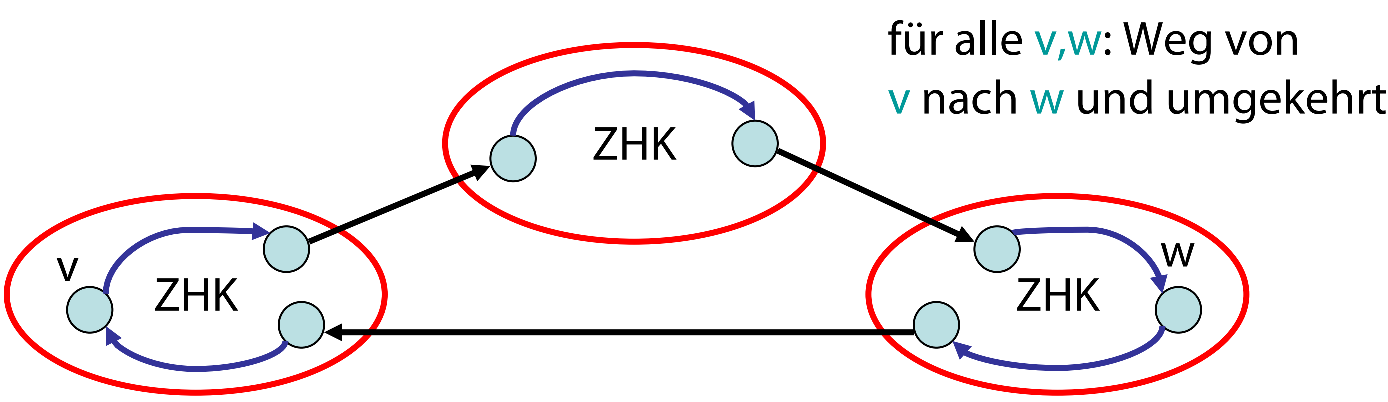


Starke ZHKs

Warum ZHKs zu einer zusammenfassbar?



Grund:



Starke ZHKs - Beispiel

Problem: wie fasst man ZHKs effizient zusammen?

Starke ZHKs

Definition:

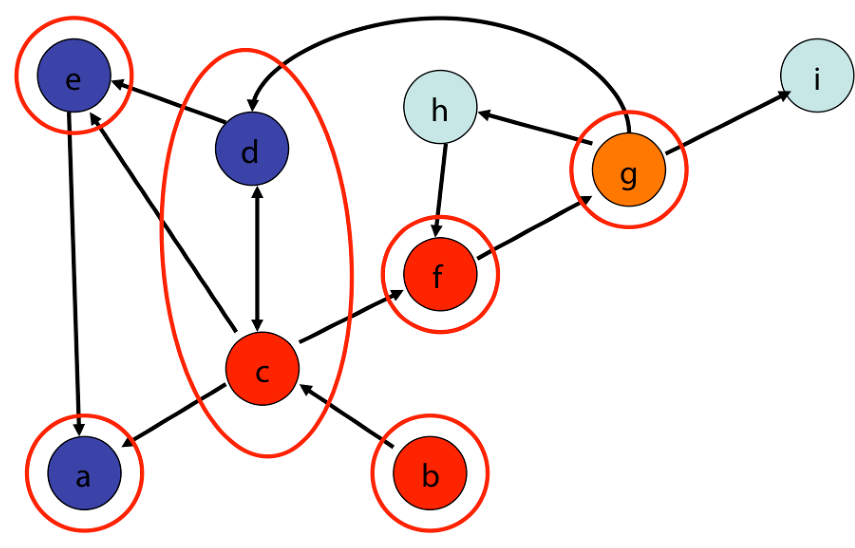
- ● ●: unfertige Knoten
- ●: fertiger Knoten
- Eine ZHK in G heißt offen, falls sie noch unfertige Knoten enthält. Sonst heißt sie (und ihre Knoten) geschlossen.
- Repräsentant einer ZHK: Knoten mit kleinster dfsNum.

Starke ZHKs

Beobachtungen (Invarianten):

1. Alle Kanten aus geschlossenen Knoten führen zu geschlossenen Knoten.
2. Der Pfad zum aktuellen Knoten enthält die Repräsentanten aller offenen ZHKs.
3. Betrachte die Knoten in offenen ZHKs sortiert nach DFS-Nummern:

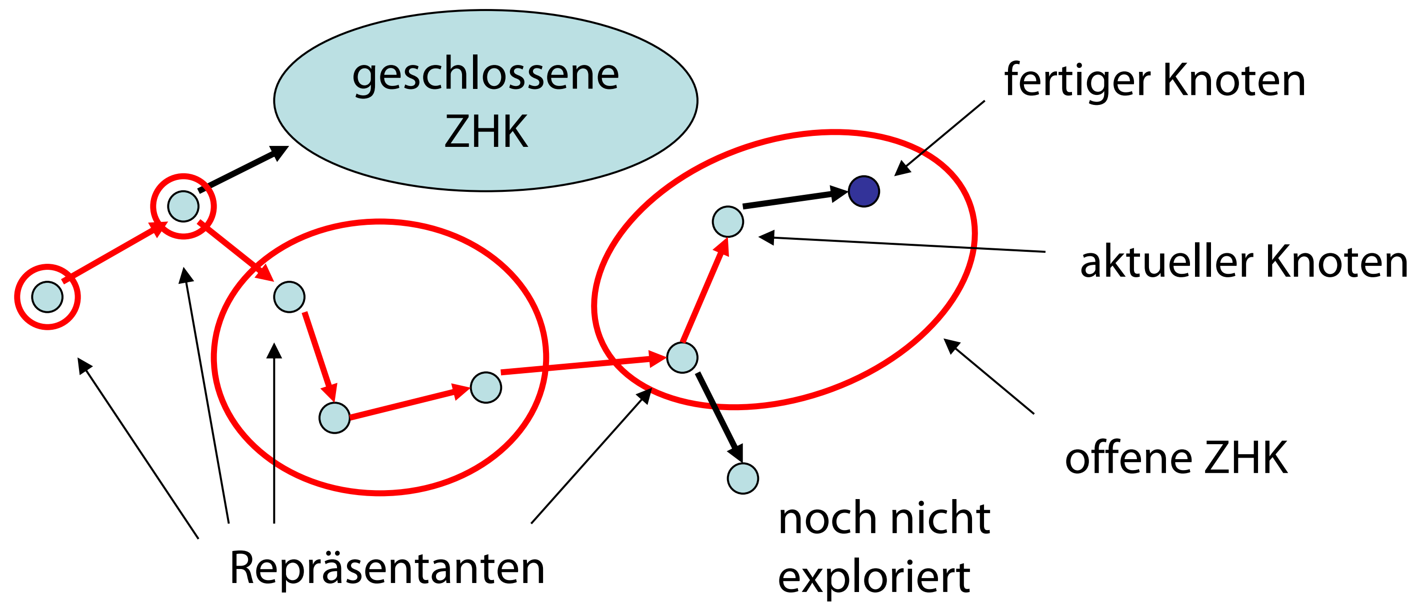
   Die Repräsentanten partitionieren diese Folge in die offenen ZHKs.



Starke ZHKs

Beweis über vollständige Induktion.

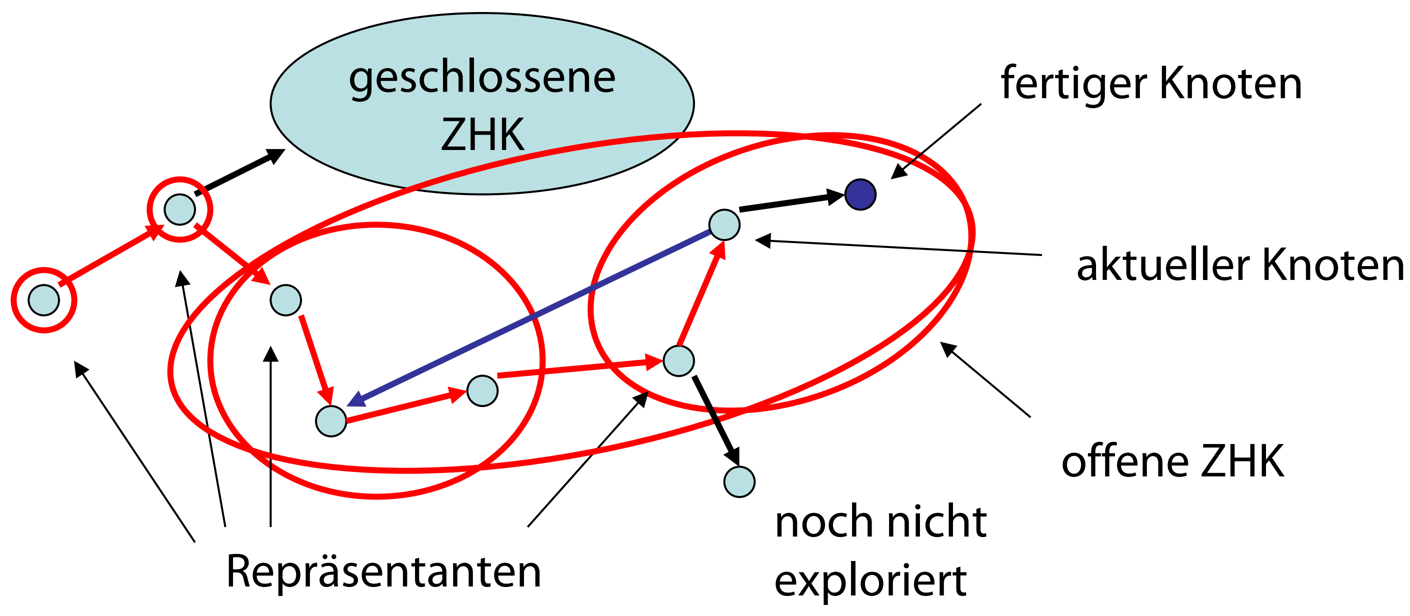
- Anfangs gelten alle Invarianten
- Wir betrachten verschiedene Fälle



Starke ZHKs

Beweis über vollständige Induktion.

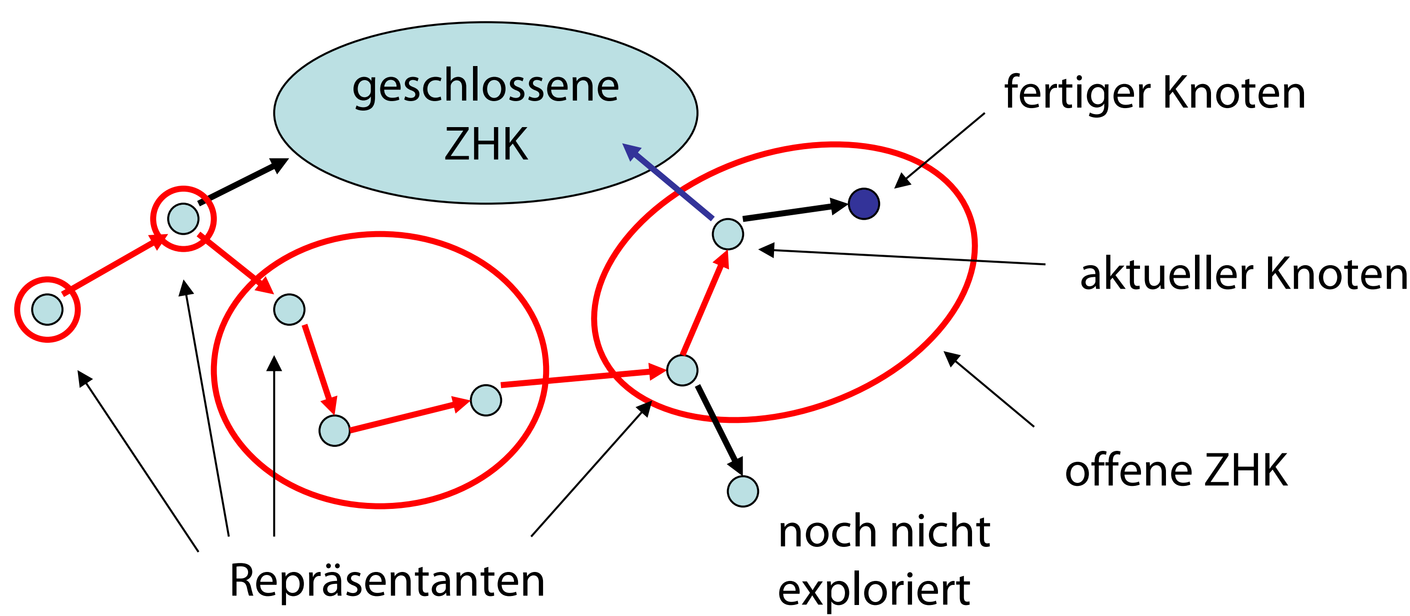
- Anfangs gelten alle Invarianten
- Fall 1: Kante zu unfertigem Knoten



Starke ZHKs

Beweis über vollständige Induktion.

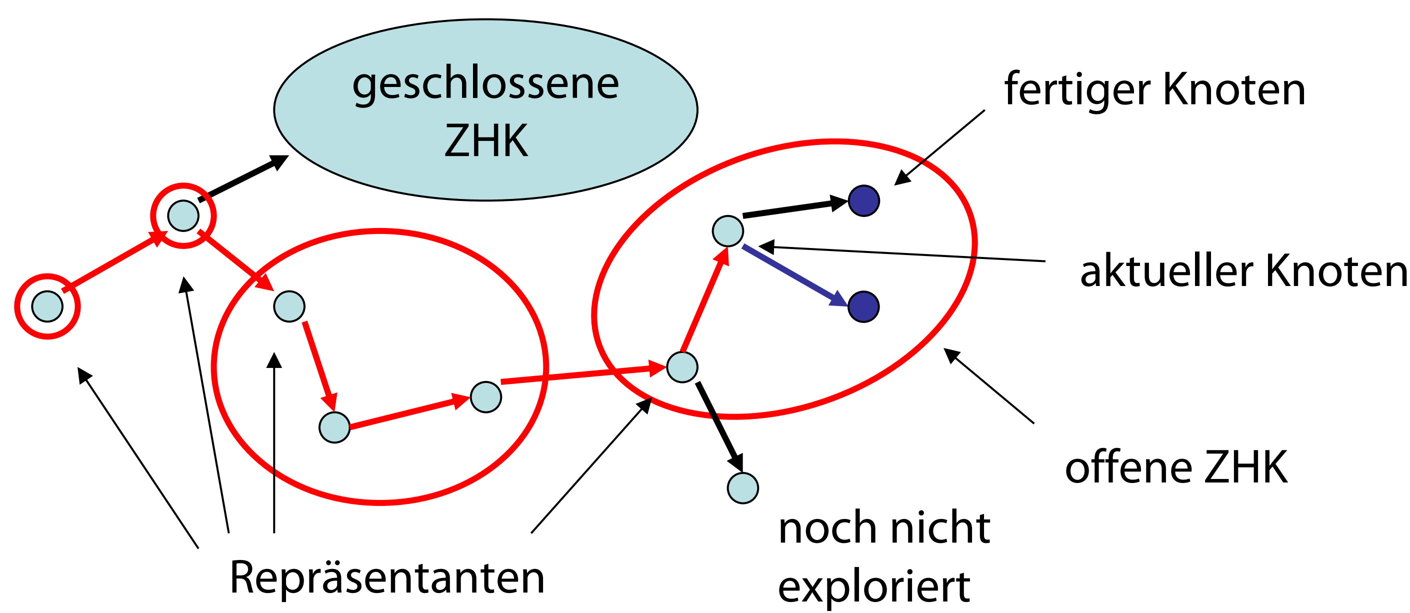
- Anfangs gelten alle Invarianten
- Fall 2: Kante zu geschlossenem Knoten



Starke ZHKs

Beweis über vollständige Induktion.

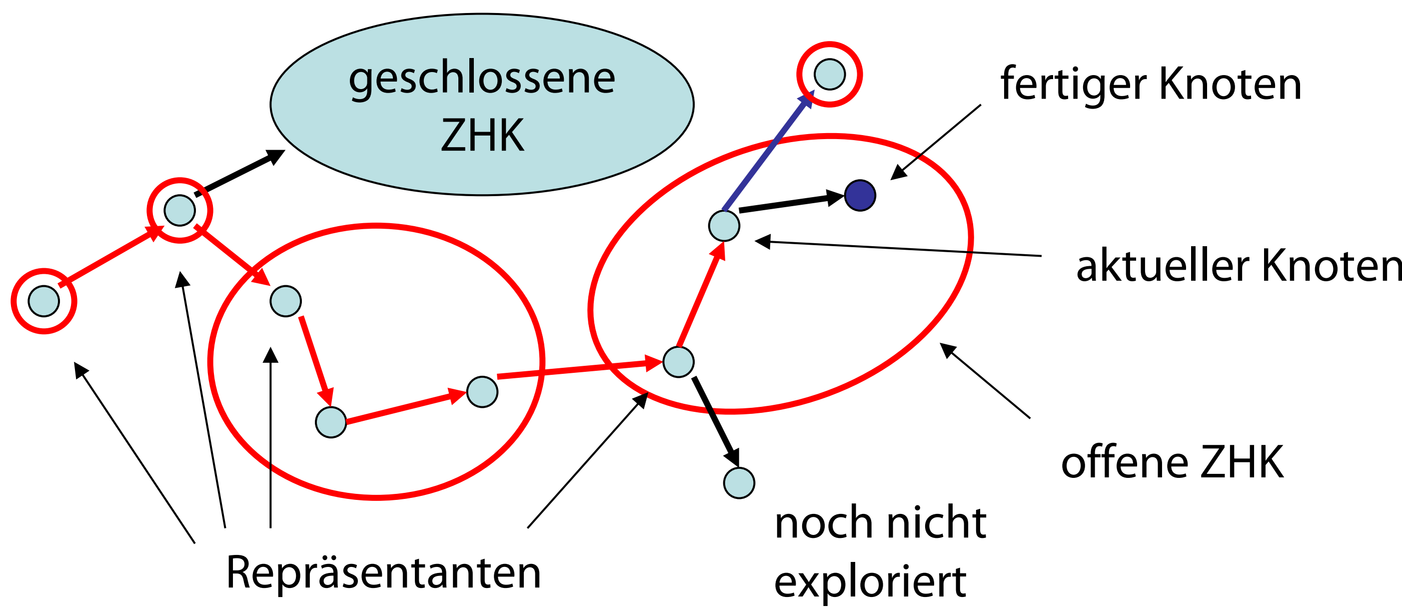
- Anfangs gelten alle Invarianten
- Fall 3: Kante zu fertigem Knoten



Starke ZHKs

Beweis über vollständige Induktion.

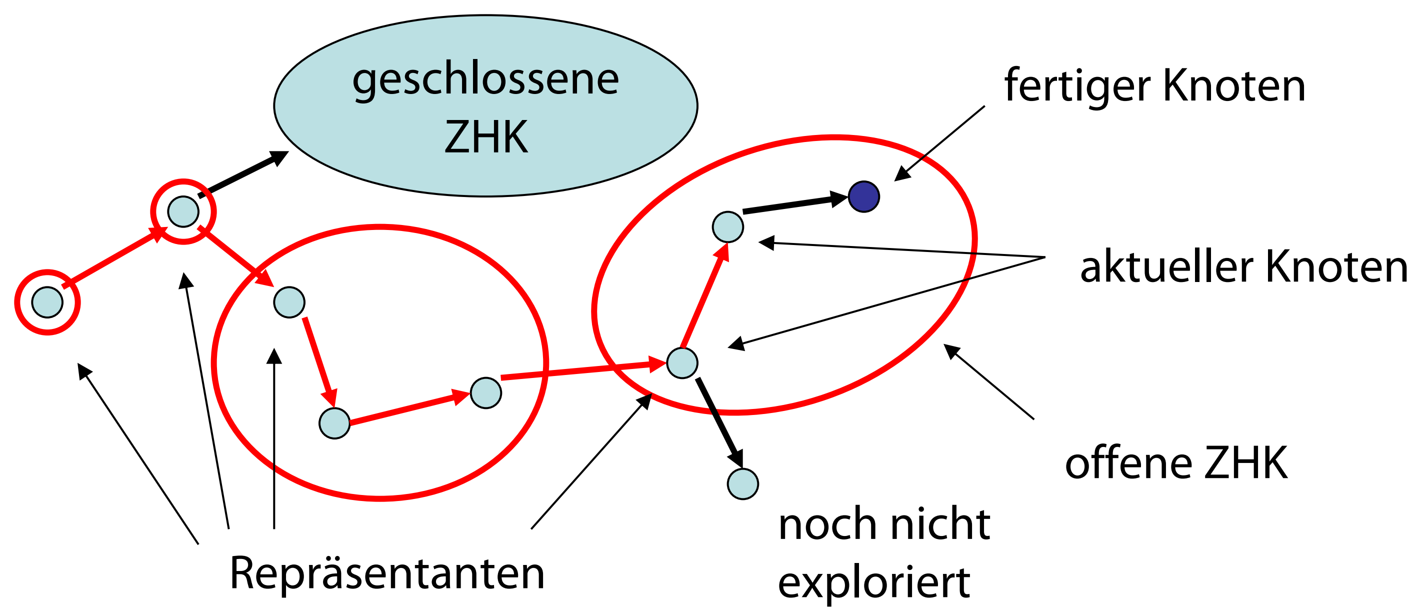
- Anfangs gelten alle Invarianten
- Fall 4: Kante zu nicht exploriertem Knoten



Starke ZHKs

Beweis über vollständige Induktion.

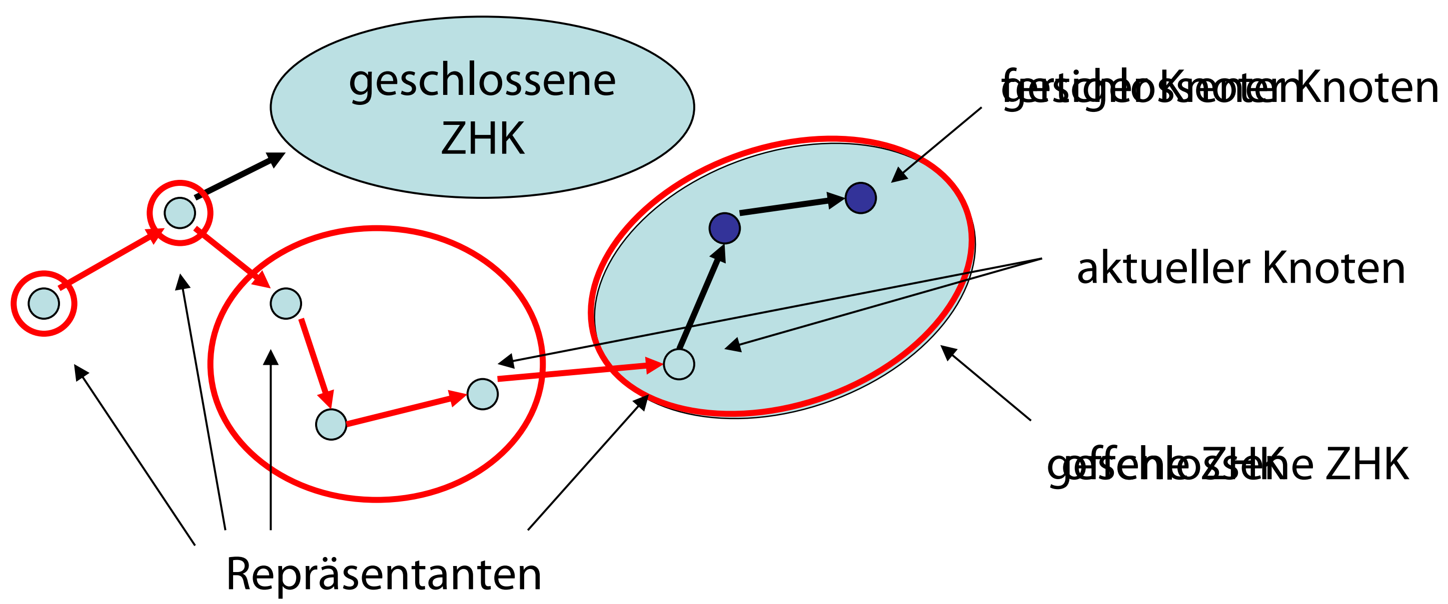
- Anfangs gelten alle Invarianten
- Fall 5: Knoten exploriert



Starke ZHKs

Beweis über vollständige Induktion.

- Anfangs gelten alle Invarianten
- Fall 5: Knoten exploriert



Starke ZHKs

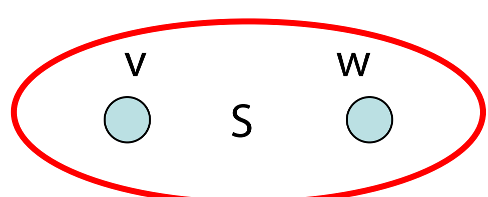
Behauptung: Eine *geschlossene* ZHK G_c im besuchten Teilgraphen C von G ist eine ZHK in G (die geschlossene ZHK G_c ist maximal)

Überlegung:

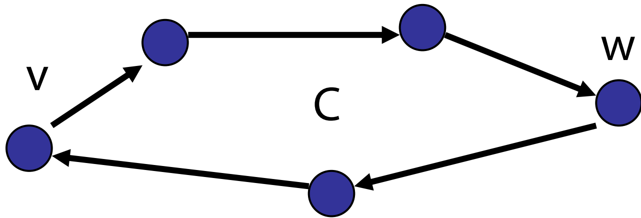
- v: geschlossener Knoten
- S: ZHK in G, die v enthält
- S_c: ZHK in G_c, die v enthält
- Es gilt: $S_c \subseteq S$ (v erreicht jeden Knoten in der ZHK)
- Zu zeigen (Maximalität): $S \subseteq S_c$

Starke ZHKs

Begründung für $S \subseteq S_c$:



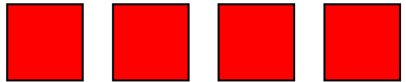
- w: beliebiger Knoten in S
- Es gibt gerichteten Kreis C durch v und w
- Nutze Invariante 1: alle Knoten in C geschlossen



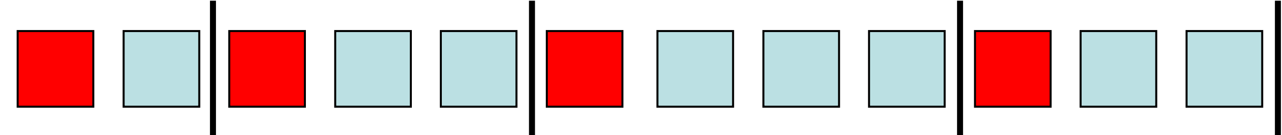
- Da alle Kanten geschlossener Knoten exploriert worden sind, ist C in G_c und daher $w \in S_c$

Invarianten 2 und 3

- Der Pfad zum aktuellen Knoten enthält die Repräsentanten aller offenen ZHKs (oReps)



- Betrachte die Knoten in offenen ZHKs sortiert nach DFS-Nummern. Die Repräsentanten partitionieren diese Folge in die offenen ZHKs (oNodes)



Stack ausreichend für beide Folgen
(oNodes sei allerdings ein Stack mit Element-Test)

Wiederholung: Tiefensuche-Schema

```
Übergeordnete Prozedur = Bestimme ZHKs
    unmark all nodes
    init()
    for s∈V do  // stelle sicher, dass alle Knoten besucht werden
       if s is not marked then
          mark s
          root(s)
          DFS(s,s)    // s: Startknoten

Procedure DFS(u,v: Node)  // u: Vater von v
    for (v,w)∈E do
       if w is marked then handleNonTreeEdge(v,w)
       else traverseTreeEdge(v,w)
          mark w
          DFS(v,w)
    backtrack(u,v)
```

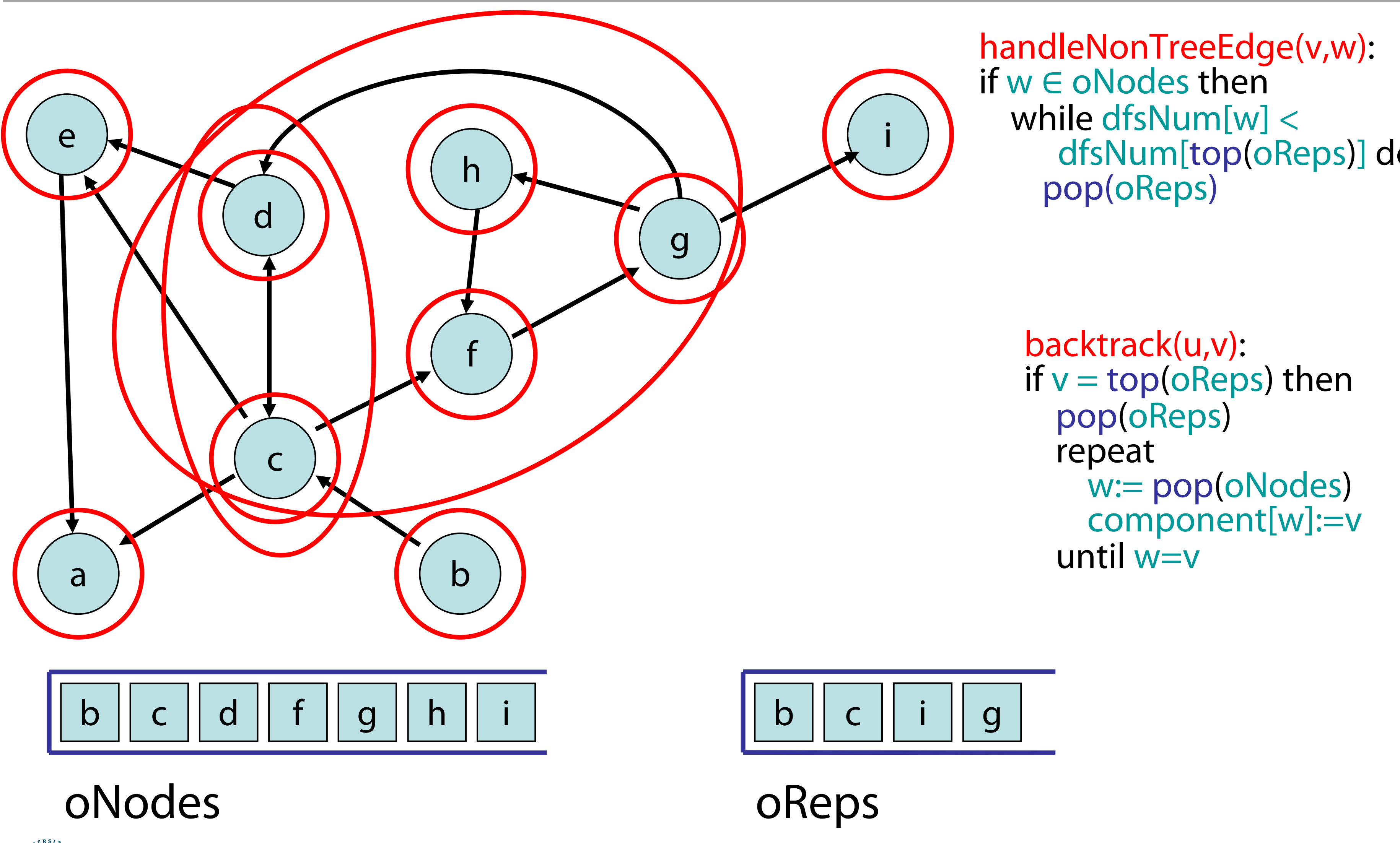
Prozeduren in rot: noch zu spezifizieren

Starke ZHKs

```
init():
   component: Array [1..n] of NodeId
   oReps = <>: Stack of NodeId
   oNodes = <>: Stack of NodeId
   dfsPos:=1

root(w) oder traverseTreeEdge(v,w):
   push(w, oReps)     // neue ZHK
   push(w, oNodes)   // neuer offener Knoten
   dfsNum[w]:=dfsPos; dfsPos:=dfsPos+1
```

Starke ZHKs - Beispiel



Starke ZHKs

Behauptung: Der DFS-basierte Algorithmus für starke ZHKs benötigt O(n+m) Zeit.

Beweis:

- init, root, traverseTreeEdge: Zeit O(1)
- backtrack, handleNonTreeEdge: da jeder Knoten nur höchstens einmal in oReps und oNodes landet, insgesamt Zeit O(n)
- DFS-Gerüst: Zeit O(n+m)

Wie realisieren wir einen Stack mit Element-Test, der in O(1) läuft?

Zusammenfassung

- Traversierung: Tiefensuche Design Pattern
- Gerichtete azyklische Graphen (DAGs)
- Starke Zusammenhangskomponenten (ZHKs)

- Im nächsten Teil:
  - Kürzeste Wege: Single-Source Shortest Paths