

Algorithmen und Datenstrukturen

Sommersemester 2020

Stefan Näher

Universität Trier

naeher@uni-trier.de

Vorlesung 16

18. Juni 2020

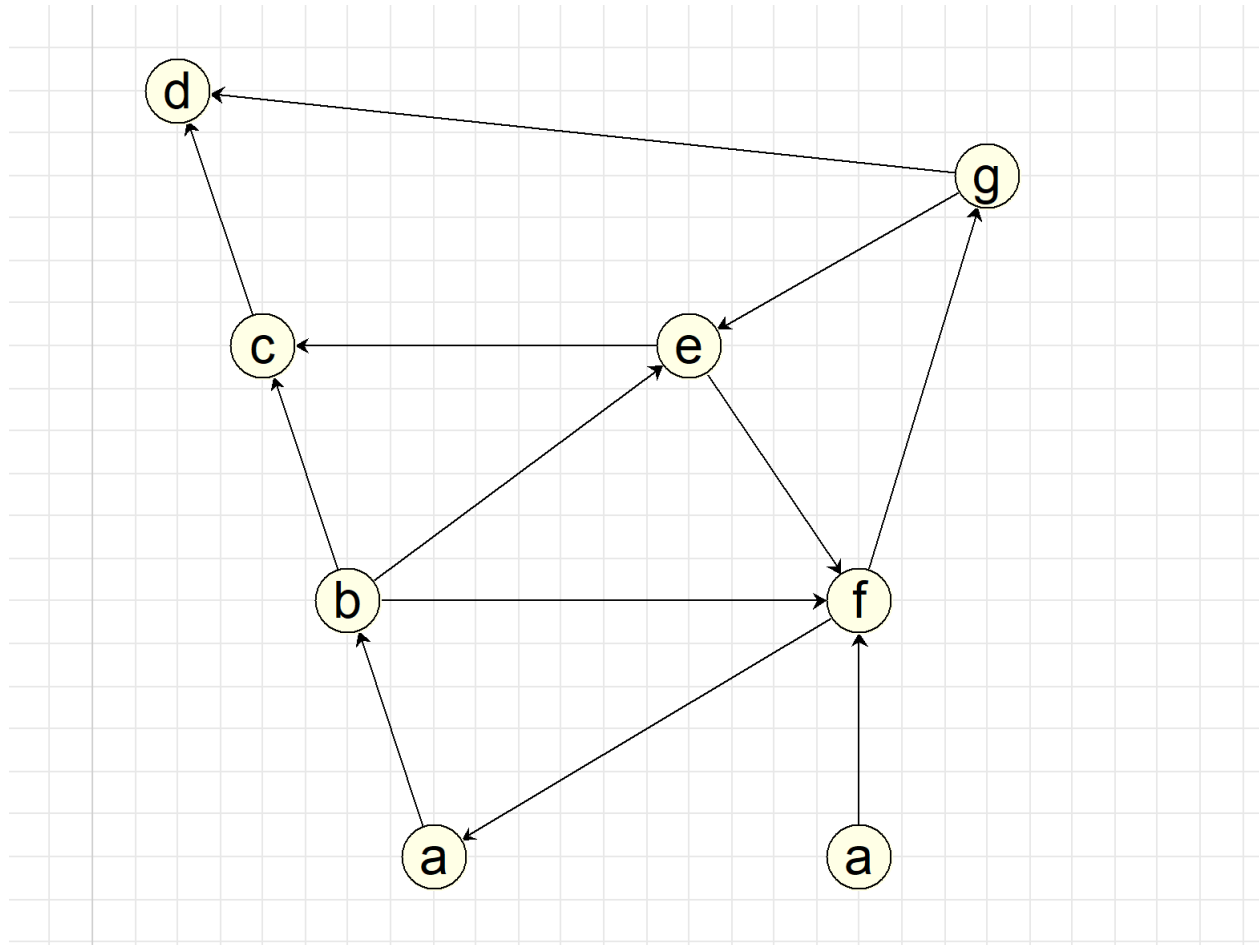
DFS Klassifizierung der Kanten

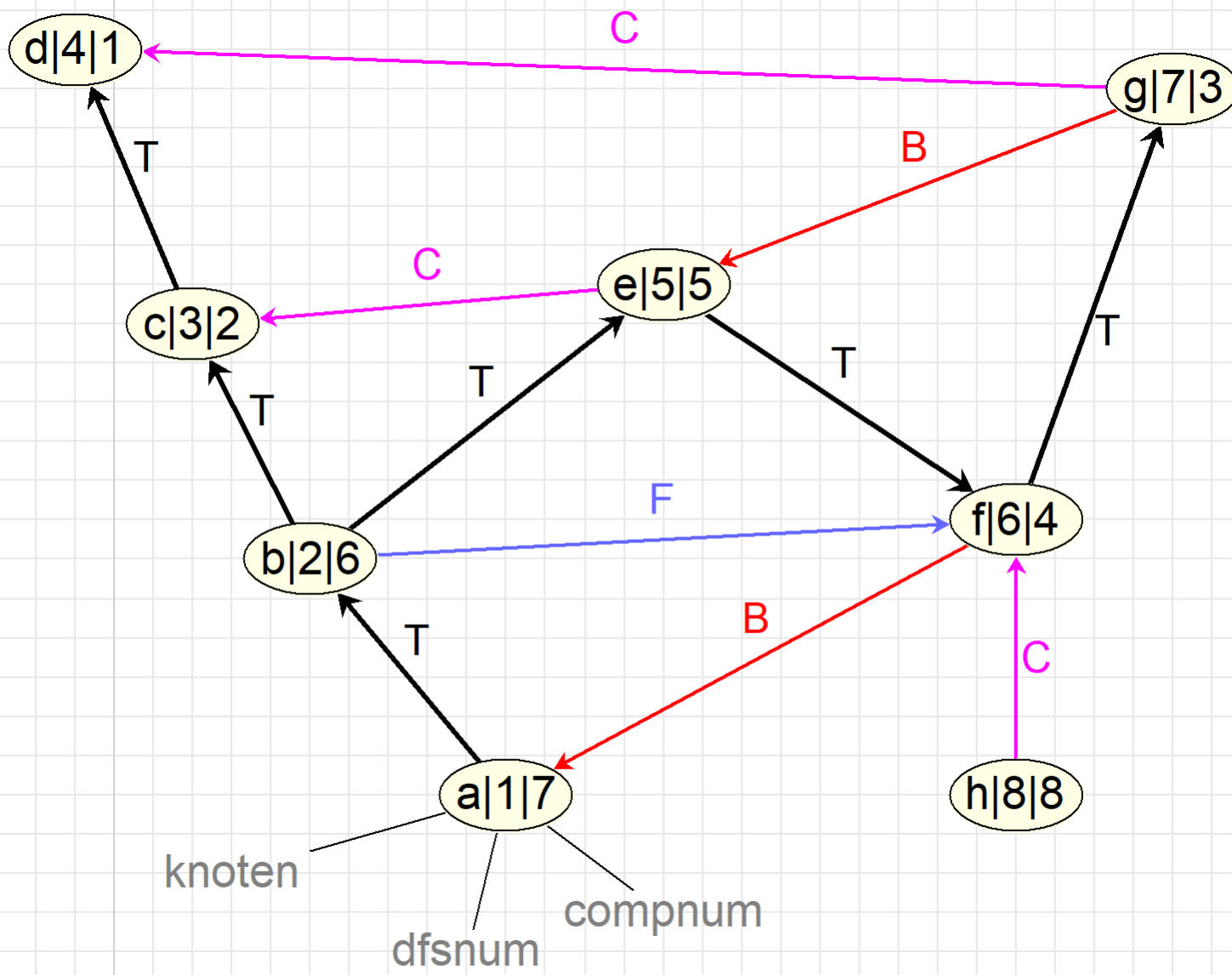
Lemma 2:

Sei $G = (V, E)$ ein gerichteter Graph und T, F, B, C , dfsnum , compnum durch einen DFS-Lauf auf G berechnet, dann gilt:

- a) T, F, B, C bilden eine Partition der Kantenmenge E
- b) T entspricht dem Baum der rekursiven dfs-Aufrufe.
- c) $v \xrightarrow[T]{*} w \iff \text{dfsnum}[v] \leq \text{dfsnum}[w] \text{ und } \text{compnum}[v] \geq \text{compnum}[w]$
- d) $(v, w) \in T \cup F \iff \text{dfsnum}[v] < \text{dfsnum}[w]$
- e) $(v, w) \in B \iff \text{dfsnum}[v] \geq \text{dfsnum}[w] \text{ und } \text{compnum}[v] \leq \text{compnum}[w]$
- f) $(v, w) \in C \iff \text{dfsnum}[v] > \text{dfsnum}[w] \text{ und } \text{compnum}[v] > \text{compnum}[w]$

Ein Beispiel





Folgerungen

Sei $G = (V, E)$ ein **azyklischer Graph**, dann gilt

1. Es gibt keine Rückwärtskanten, d.h. $B = \emptyset$.
 2. Aus Lemma 2 folgt dann: $\text{compnum}(v) > \text{compnum}(w)$ für alle $(v, w) \in E$.
 3. Dann ist $\text{ord} : V \longrightarrow \{1, \dots, n\}$ mit $\text{ord}(v) = n - \text{compnum}(v) + 1$ eine topologische Sortierung von G .
- alternativer Algorithmus zur Berechnung einer topologischen Sortierung mit Laufzeit $\mathcal{O}(n + m)$.

Eine Anwendung von DFS

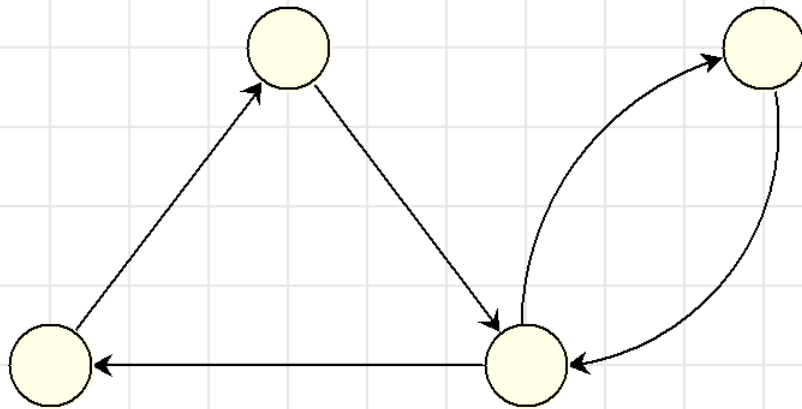
Die Berechnung der starken Zusammenhangskomponenten

Definition:

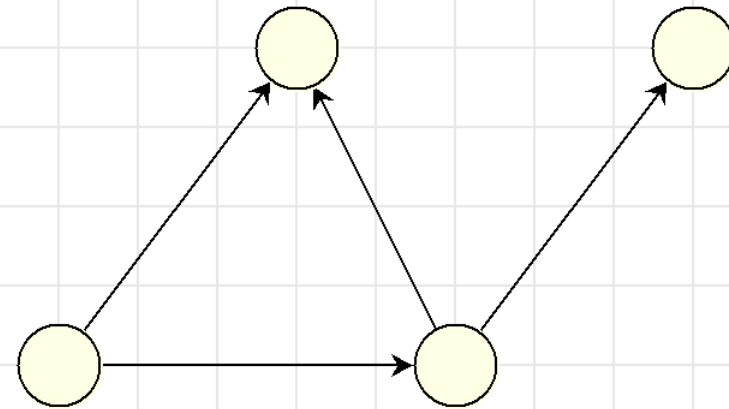
1. Ein gerichteter Graph $G = (V, E)$ ist **stark zusammenhängend**, wenn es für alle Knoten $v, w \in V$ einen Pfad von v nach w gibt d.h. $v \xrightarrow{*} w$
2. Die **stark zusammenhängenden Komponenten** (SZK) von G sind die maximalen stark zusammenhängenden Teilgraphen von G .

Beispiel: stark zusammenhängend

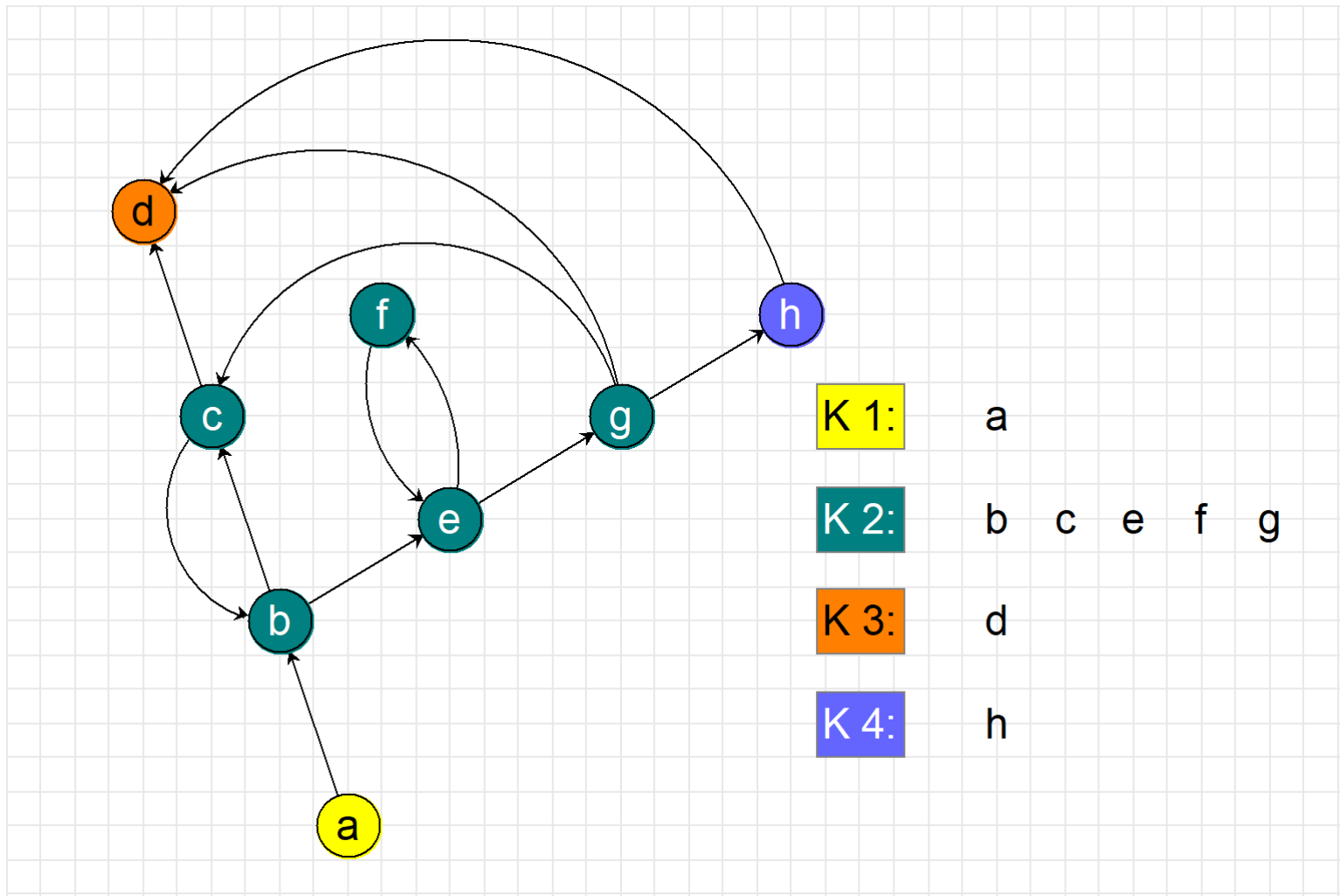
stark zusammenhängend



nicht stark zus.haengend



Beispiel: SZKs



Idee zur Berechnung der SZK

Führe DFS auf G aus. Sei $G' = (V', E')$ der Teilgraph von G , der von den benutzten Kanten E' aufgespannt wird.

Wir verwalten die SZK von G'

Im Beispiel:

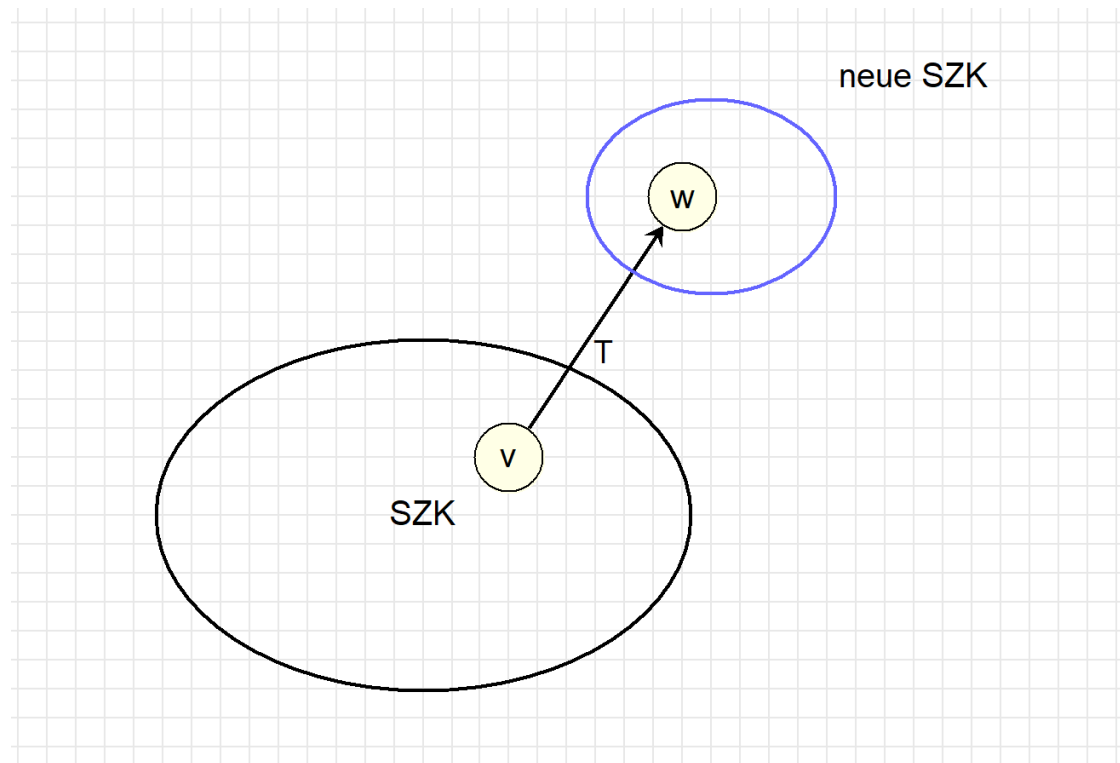
Initialisierung:

$$V' = \{ a \}, E' = \emptyset, SZK = \{ \{ a \} \}$$

Sei (v, w) die nächste von DFS betrachtete Kante ($v \in V'$).

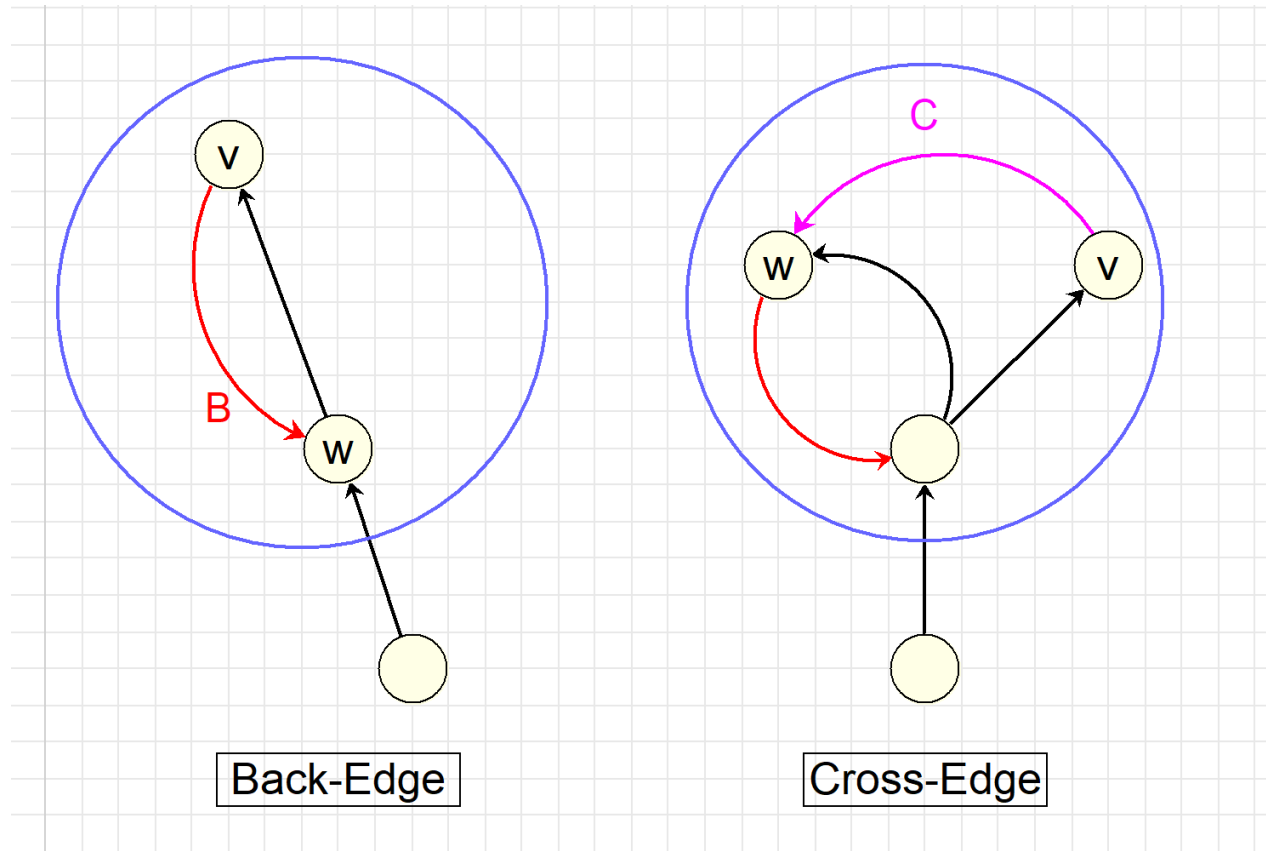
Fall 1: $(v, w) \in T$

Füge eine neue SZK hinzu, die nur aus dem Knoten w besteht



Fall 2: $(v, w) \notin T$

dann kann (v, w) mehrere SZKs zu einer zusammen mischen

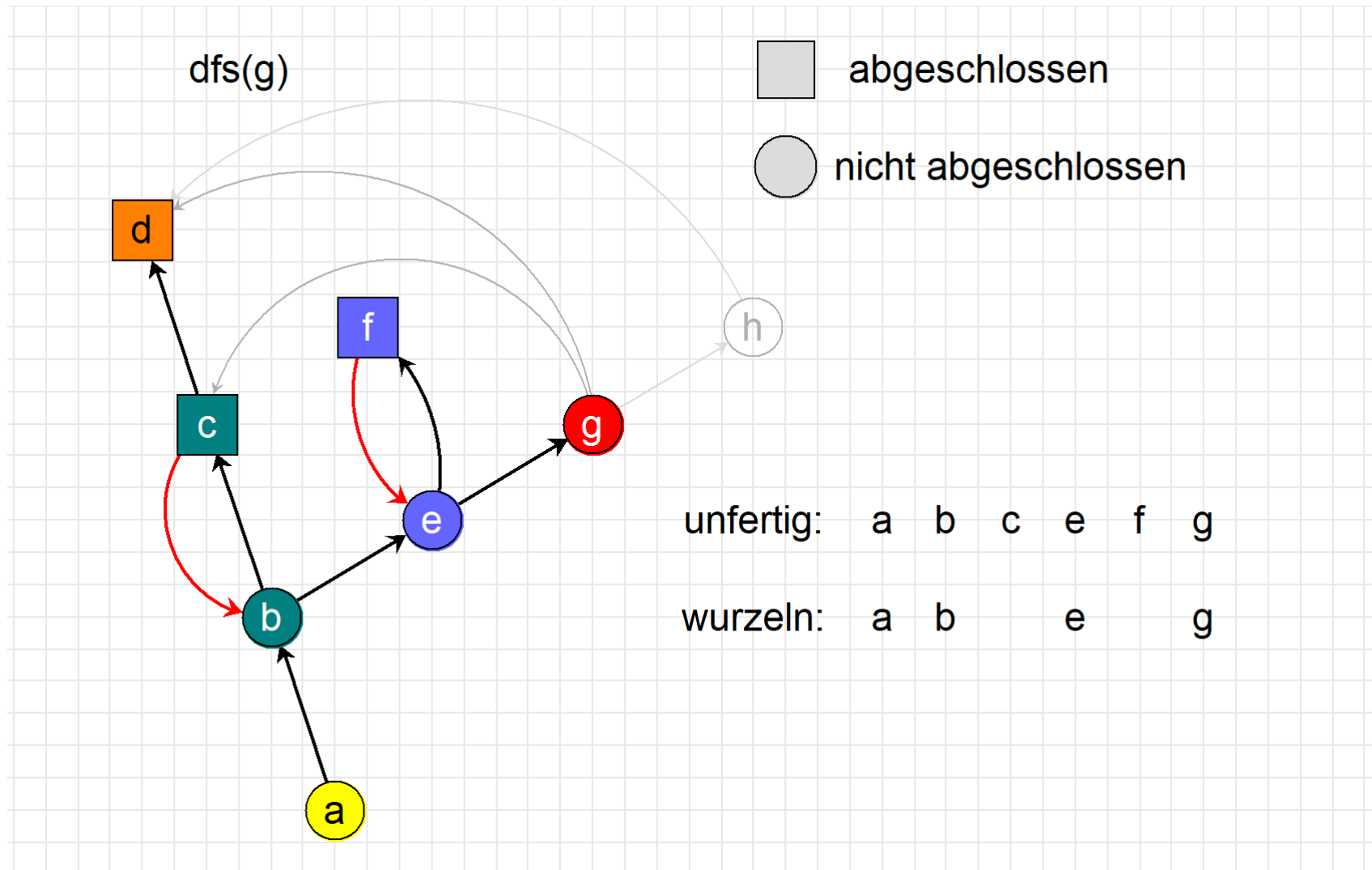


Problem

Wie kann man den Misch-Schritt effizient implementieren ?

Einige Definitionen und Bezeichnungen

1. Eine SZK K heißt **abgeschlossen**, falls die DFS-Aufrufe $\text{dfs}(v)$ für alle $v \in K$ abgeschlossen sind.
2. Die **Wurzel** einer SZK K ist der Knoten mit der kleinsten dfsnum in K .
3. **unfertig** = Folge der Knoten, für die dfs aufgerufen wurde, aber deren SZK noch nicht abgeschlossen ist, nach dfsnum sortiert.
4. **wurzeln** = Folge der Wurzeln der noch nicht abgeschlossenen SZKs, nach dfsnum sortiert (Unterfolge von unfertig).



Beobachtungen

1. $v \in \text{unfertig} \iff v \xrightarrow{*} g$
2. $\nexists (v, w) \in E$ mit v in abgeschlossener und w in nicht-abgeschlossener Komponente.

Fortsetzung des Beispiels: Wir betrachten die Kanten aus dem Knoten g .

$(g, d) \in C$

es passiert nichts, da d in abgeschlossener SZK
d.h. (g, d) kann keinen Kreis schließen.

$(g, c) \in C$

vereinigt die 3 SZKs mit den Wurzeln b, e, g durch Streichen von e und g aus wurzeln.

unfertig: a b c e f g

wurzeln: a b

$(g, h) \in T$

Erzeuge neue SZK { h } durch Anfügen von h ans Ende von unfertig und ans Ende von wurzeln

unfertig: a b c e f g h

wurzeln: a b h

Rückkehr aus einem dfs-Aufruf

Bei der Rückkehr aus einem Aufruf $\text{dfs}(v)$ wird getestet, ob v eine Wurzel ist (rechtestes Element der *wurzel*-Folge). Falls ja, wird die Komponente mit Wurzel v ausgegeben und aus *unfertig* und *wurzeln* entfernt.

Beachte:

Hinzufügen in und Streichen aus den *unfertig* und *wurzel*-Listen geschieht immer am rechten (oder oberen) Ende.

→ **Stack**