

Algorithmen und Datenstrukturen

Sommersemester 2020

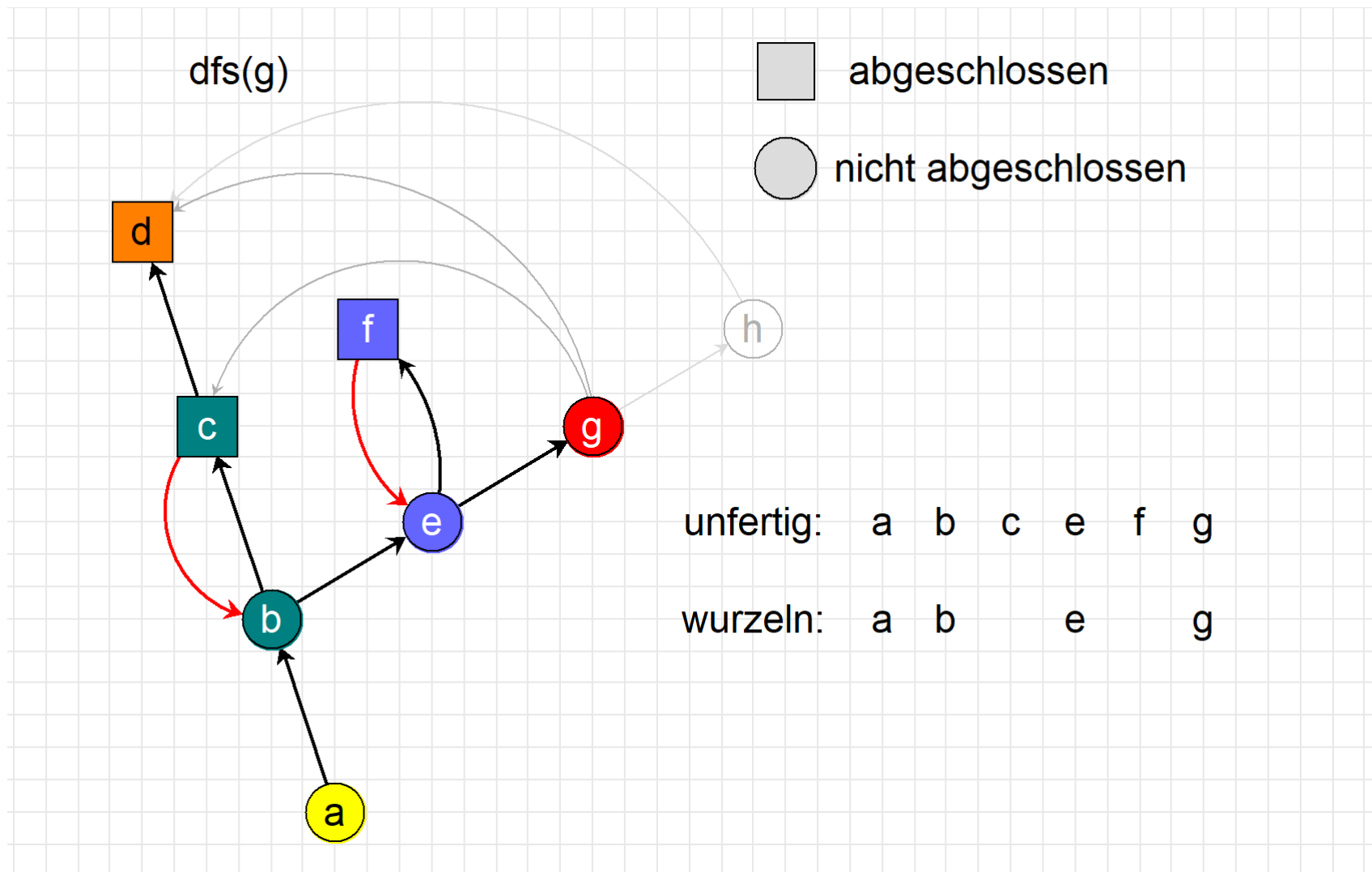
Stefan Näher

Universität Trier

naeher@uni-trier.de

Vorlesung 17

24. Juni 2020



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

Die allgemeine Situation

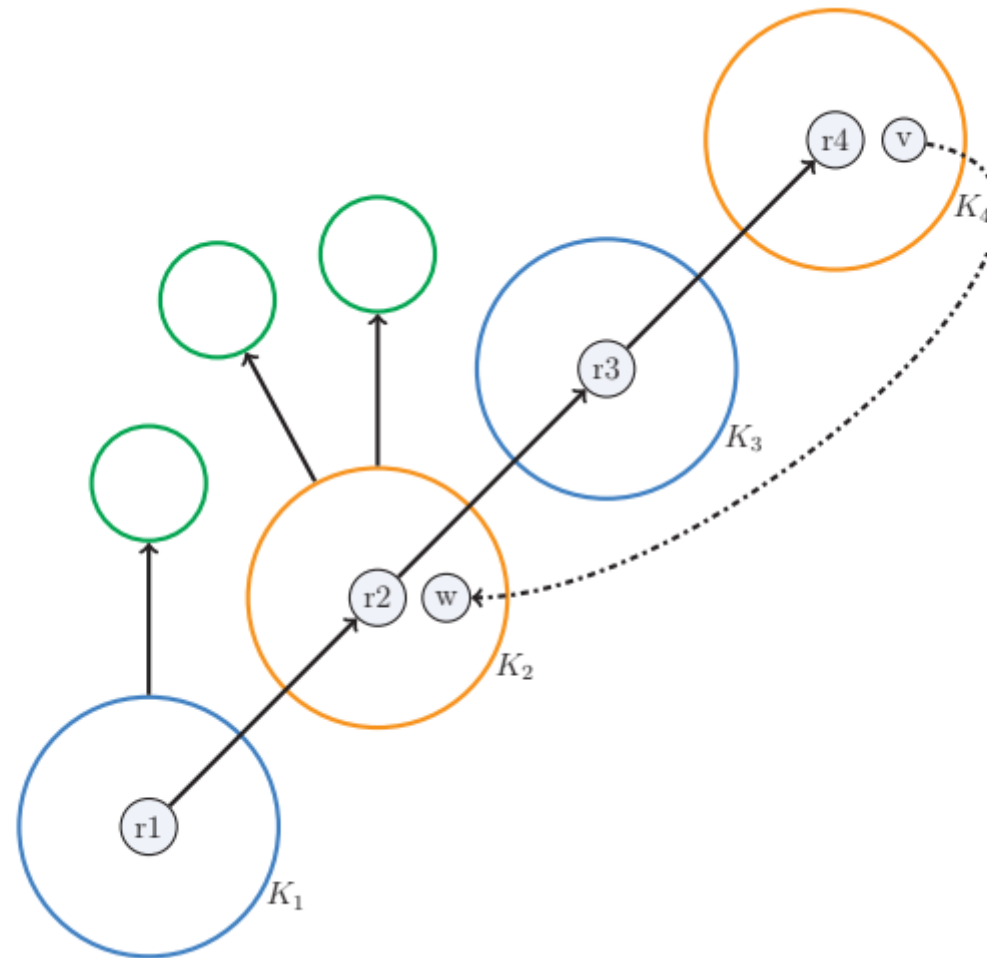
Hinzufügen zu und Streichen aus den Folgen `unfertig` und `wurzeln` geschieht immer am rechten (oder oberen) Ende.

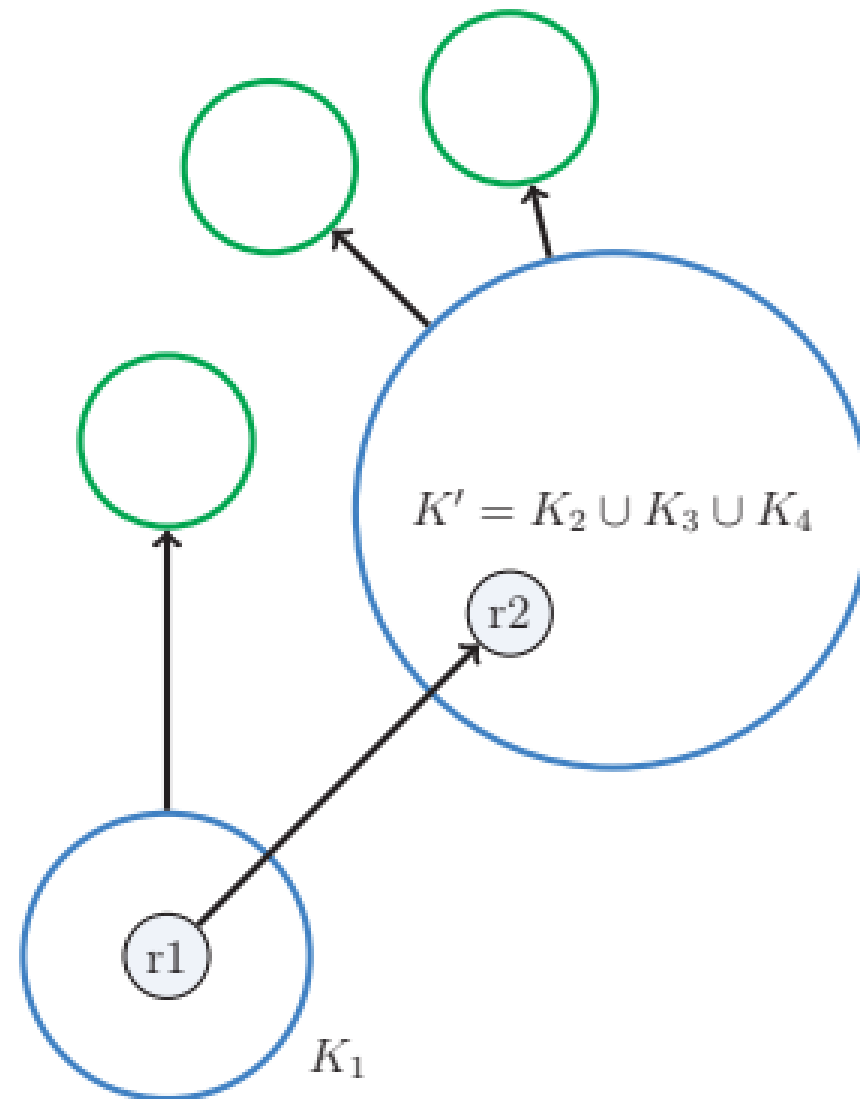
—→ Datenstruktur: **Stack**

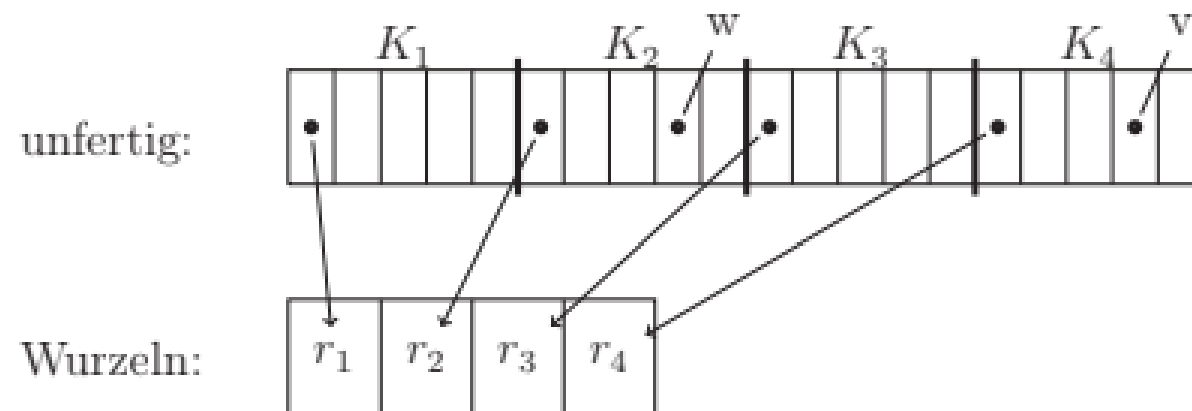
Fall 1: $(v, w) \in T$

```
wurzeln.push(w);  
unfertig.push(w);
```

Fall 2: $(v, w) \notin T$ und $w \in \text{unfertig}$







Aktion:

```

1 while(dfsnum[wurzeln.top()] > dfsnum[w]){
2     wurzeln.pop();
3 }

```

Abschluss eines dfs-Aufrufs

Bei der Rückkehr aus einem Aufruf $\text{dfs}(v)$ testen wir, 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.

```
if  $v = \text{wurzeln.top}()$  then  
    repeat  $w \leftarrow \text{unfertig.pop}();$   
        PRINT( $w$ );  
    until  $v = w;$   
     $\text{wurzeln.pop}();$   
fi
```

Die modifizierte dfs-Funktion

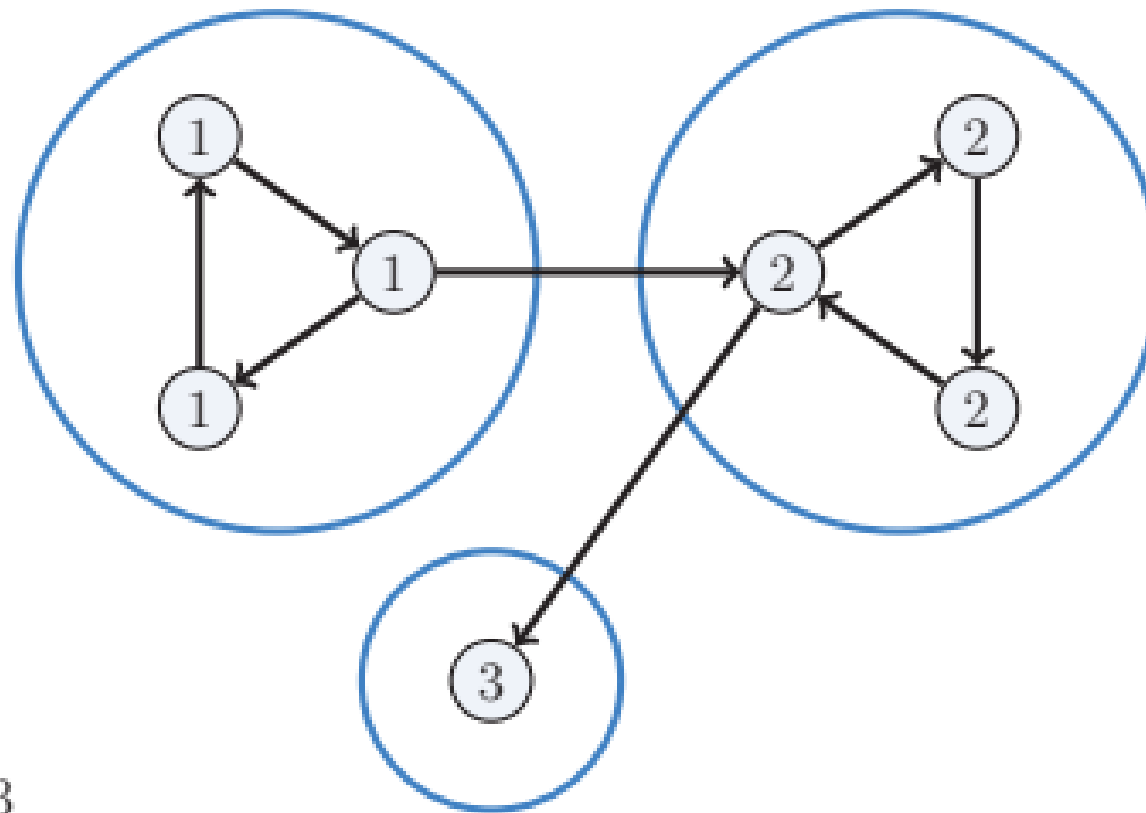
1. `dfs(v)`
2. `besucht[v]  $\leftarrow$  true;`
3. `dfsnum[v]  $\leftarrow$  ++dfs_count1;`
4. `unfertig.push(v);`
5. `in_unfertig[v]  $\leftarrow$  true;`
6. `wurzeln.push(v);`

```
7. forall  $w \in V$  mit  $(v, w) \in E$  do  
8.   if  $\neg \text{besucht}[w]$   
9.   then  $\text{dfs}(w)$ ;  
10.  else if  $\text{in\_unfertig}[w]$   
11.    then // Mischen  
12.      while  $\text{dfsnum}[\text{wurzeln.top}()] > \text{dfsnum}[w]$  do  
13.         $\text{wurzeln.pop}()$ ;  
14.      od  
15.    fi  
16.  fi  
17. od
```

```
18. if  $v = \text{wurzeln.top}()$  then  
19.    $\text{szk\_count}++$ ;  
20.   repeat  $w \leftarrow \text{unfertig.pop}()$ ;  
21.      $\text{in\_unfertig}[w] \leftarrow \text{false}$ ;  
22.      $\text{szk\_num}[w] \leftarrow \text{szk\_count}$ ;  
23.   until  $v = w$ ;  
24.    $\text{wurzeln.pop}()$ ;  
25. fi
```

Die Haupt-Funktion

```
1. dfs_count  $\leftarrow$  0;
2. szk_count  $\leftarrow$  0;
3. unfertig.clear();
4. wurzeln.clear();
5. forall  $v \in V$  do
6.   besucht[v]  $\leftarrow$  false;
7.   in_unfertig[v]  $\leftarrow$  false;
8. od
9. forall  $v \in V$  do
10.  if  $\neg$ besucht[v] then
11.    dfs(v);
12.  fi
13. od
```



$K = 3$

Zusammenfassung

Satz:

Die stark zusammenhängenden Komponenten eines gerichteten Graphen G mit n Knoten und m Kanten können durch **einen DFS-Lauf** in Zeit $\mathcal{O}(n+m)$ berechnet werden.