



## Datenstrukturen, Algorithmen und Programmierung 2 (DAP2)

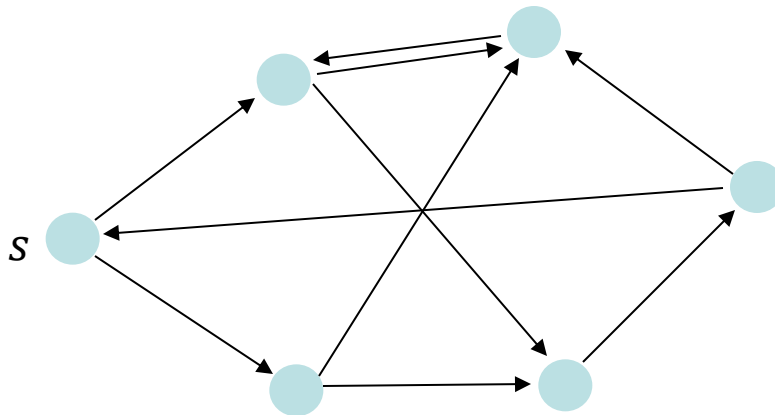
# Graphalgorithmen

## *Breitensuche*

- Durchlauf verbundenen Graph von Startknoten  $s$
- Berechne kürzeste Wege (Anzahl Kanten) von  $s$  zu anderen Knoten im Graph
- Eingabegraph in Adjazenzlistendarstellung
- Laufzeit  $\mathbf{O}(|V| + |E|)$

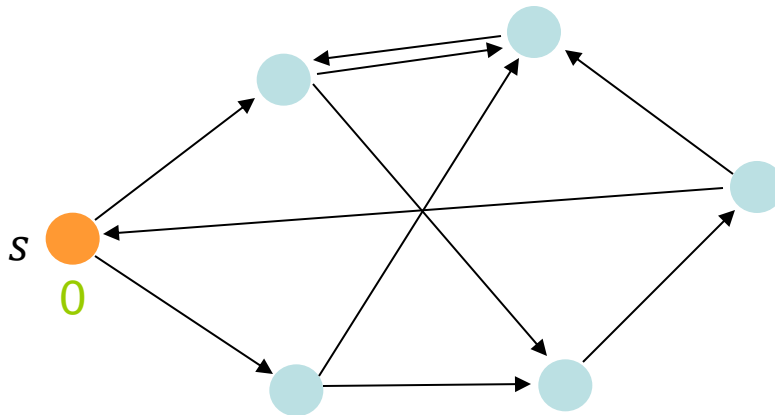
## Graphalgorithmen

### *Breitensuche*



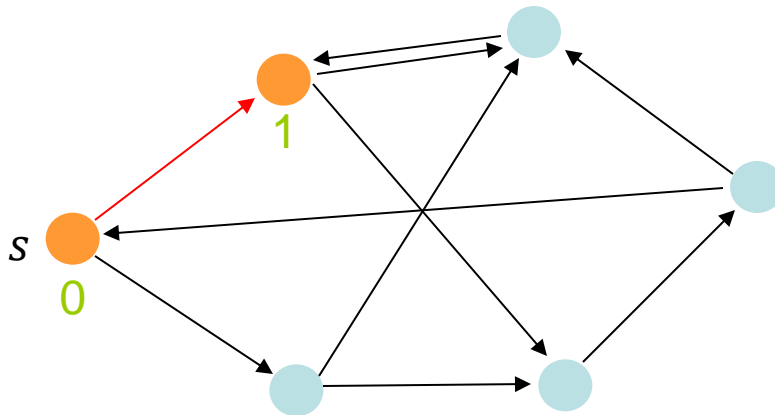
## Graphalgorithmen

### Breitensuche



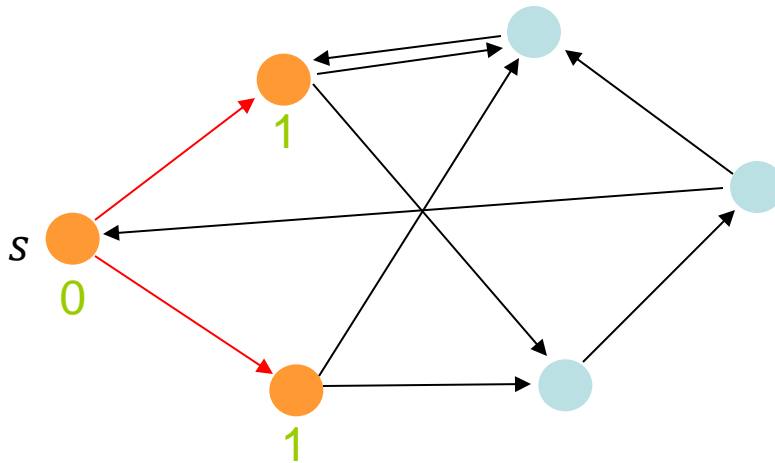
## Graphalgorithmen

### Breitensuche



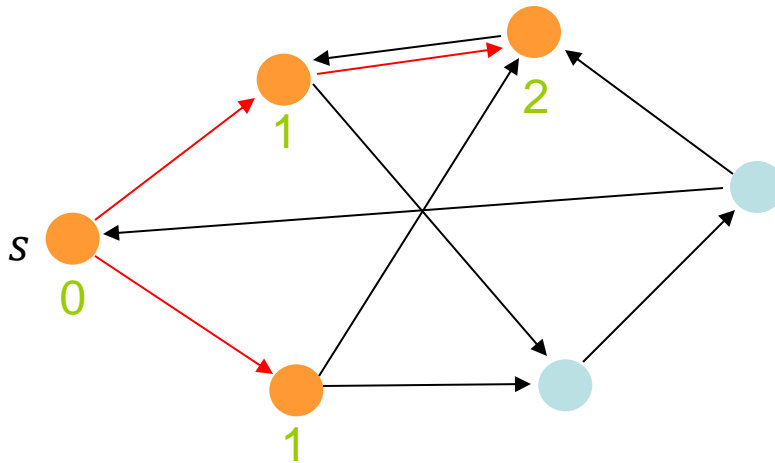
## Graphalgorithmen

### Breitensuche



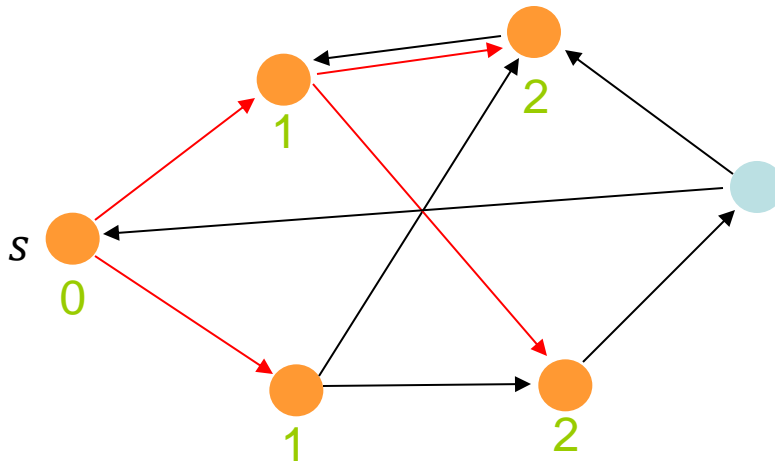
## Graphalgorithmen

### Breitensuche



## Graphalgorithmen

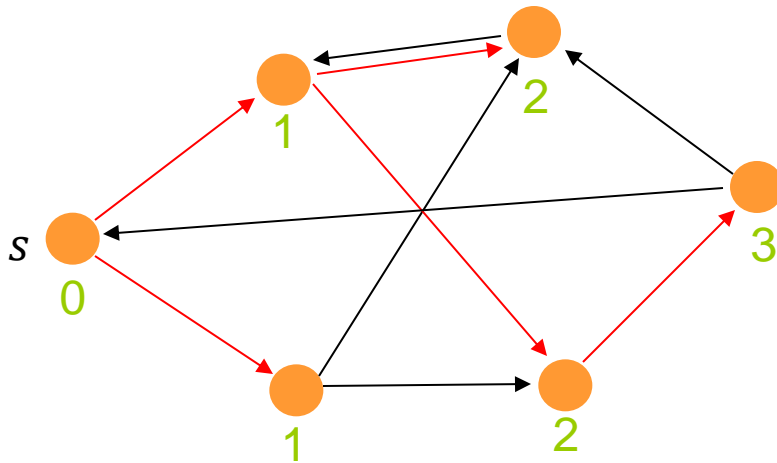
### Breitensuche





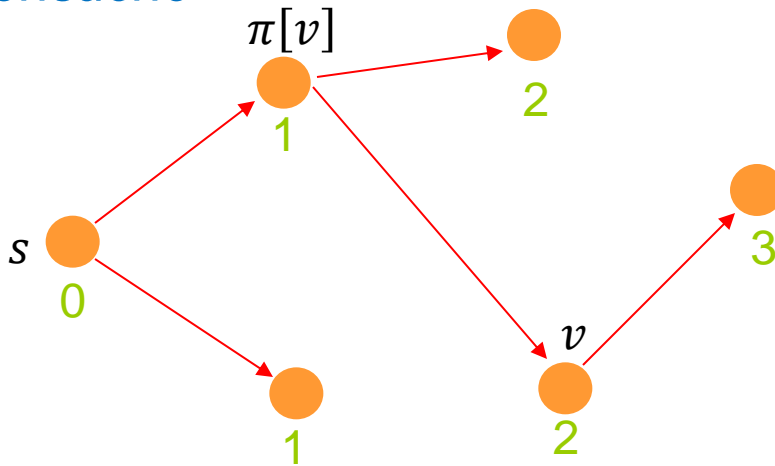
## Graphalgorithmen

### Breitensuche



## Graphalgorithmen

### Breitensuche



Vorgängergraph  $G$  enthält alle Kanten  $(\pi[v], v)$ .

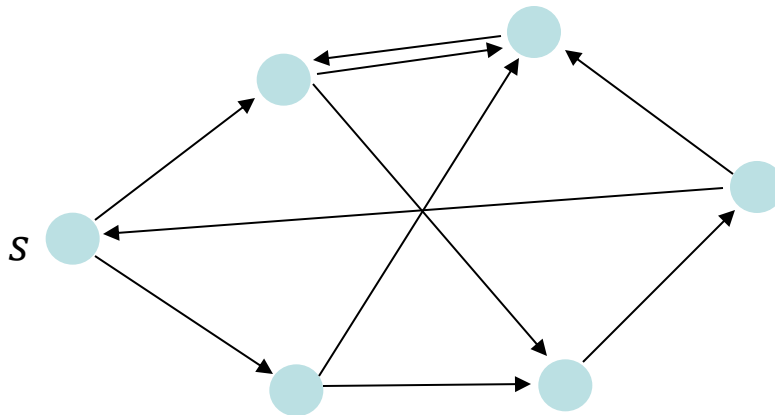
## Graphalgorithmen

### *Tiefensuche*

- Suche zunächst „tiefer“ im Graph
- Neue Knoten werden immer vom zuletzt gefundenen Knoten entdeckt
- Sind alle adjazenten Knoten des zuletzt gefundenen Knoten  $v$  bereits entdeckt, springe zurück zum Knoten, von dem aus  $v$  entdeckt wurde
- Wenn irgendwelche unentdeckten Knoten übrigbleiben, starte Tiefensuche von einem dieser Knoten

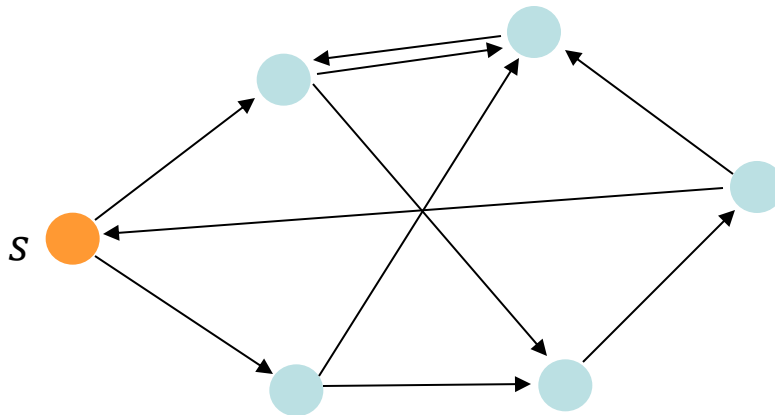
## Graphalgorithmen

### *Tiefensuche*



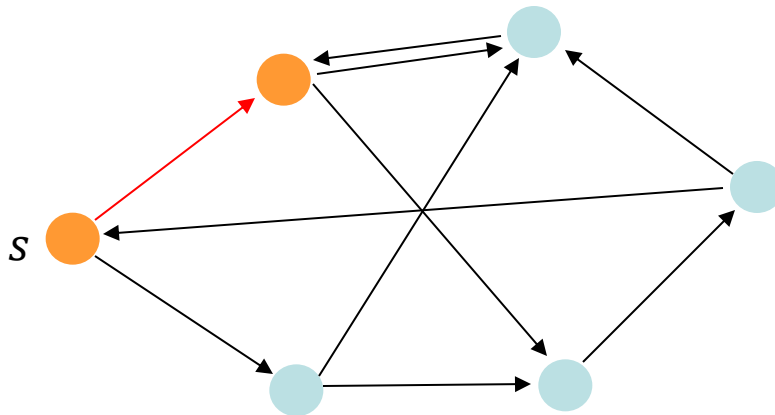
## Graphalgorithmen

### *Tiefensuche*



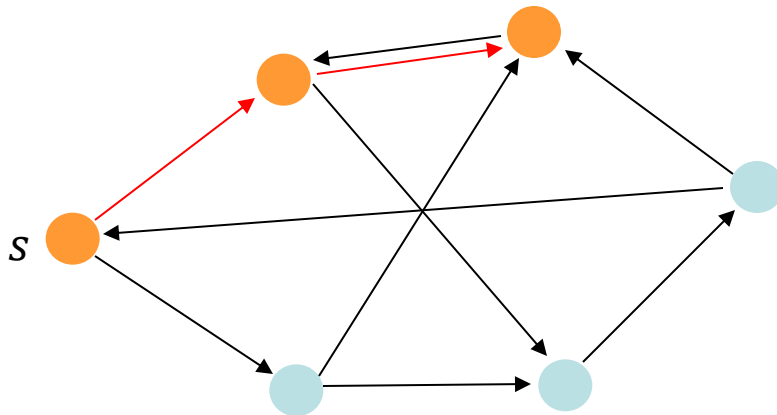
## Graphalgorithmen

### *Tiefensuche*



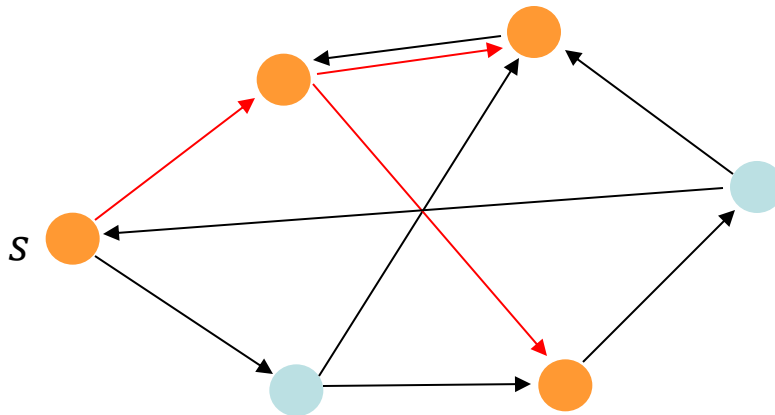
## Graphalgorithmen

### *Tiefensuche*



## Graphalgorithmen

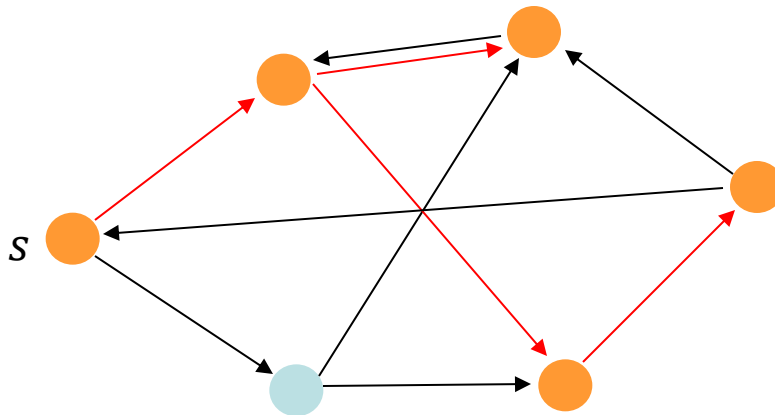
### *Tiefensuche*





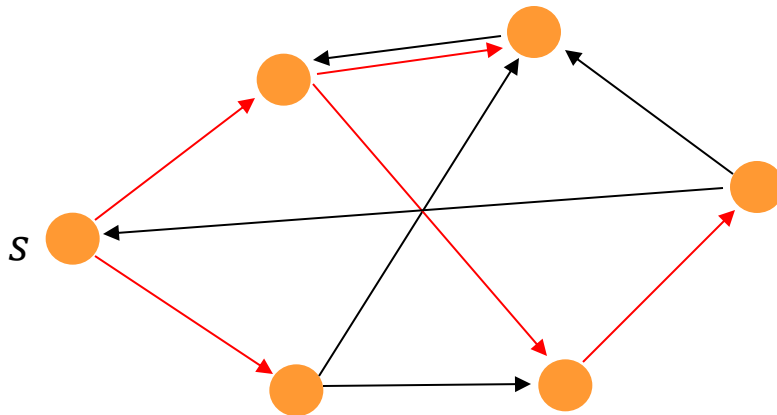
## Graphalgorithmen

### *Tiefensuche*



## Graphalgorithmen

### *Tiefensuche*



## Graphalgorithmen

### *Invariante Tiefensuche*

- Zu Beginn: alle Knoten weiß
- Entdeckte Knoten werden grau
- Abgearbeitete Knoten werden schwarz
- Zwei Zeitstempel:  $d[v]$  und  $f[v]$  (liegen zwischen 1 und  $2|V|$ )
- $d[v]$ :  $v$  ist entdeckt
- $f[v]$ :  $v$  ist abgearbeitet

## Graphalgorithmen

### *Zeitstempel der Tiefensuche*

- $d[v] < f[v]$
- Vor  $d[v]$  ist  $v$  weiß
- Zwischen  $d[v]$  und  $f[v]$  ist  $v$  grau
- Nach  $f[v]$  ist  $v$  schwarz

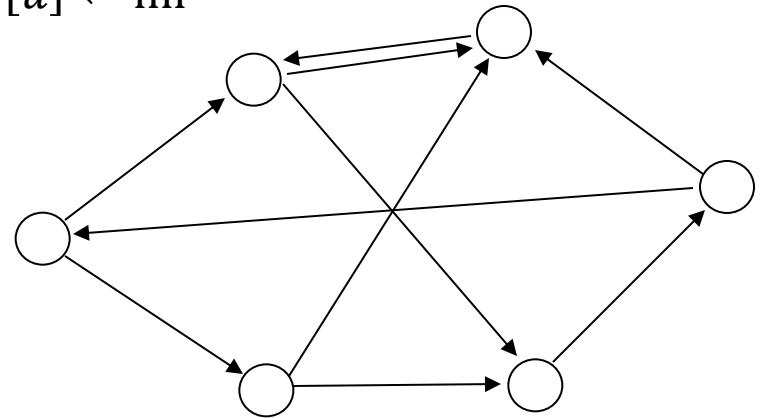
## Graphalgorithmen

DFS( $G$ )

1. **for each** vertex  $u \in V$  **do** color[ $u$ ]  $\leftarrow$  weiß;  $p[u] \leftarrow$  nil
2. time  $\leftarrow$  0
3. **for each** vertex  $u \in V$  **do**
4.   **if** color[ $u$ ] = weiß **then** DFS-Visit( $u$ )

DFS-Visit( $u$ )

1. color[ $u$ ]  $\leftarrow$  grau
2. time  $\leftarrow$  time + 1;  $d[u] \leftarrow$  time
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.   **if** color[ $v$ ] = weiß **then**  $p[v] \leftarrow u$  ; DFS-Visit( $v$ )
5. color[ $u$ ]  $\leftarrow$  schwarz
6. time  $\leftarrow$  time + 1;  $f[u] \leftarrow$  time



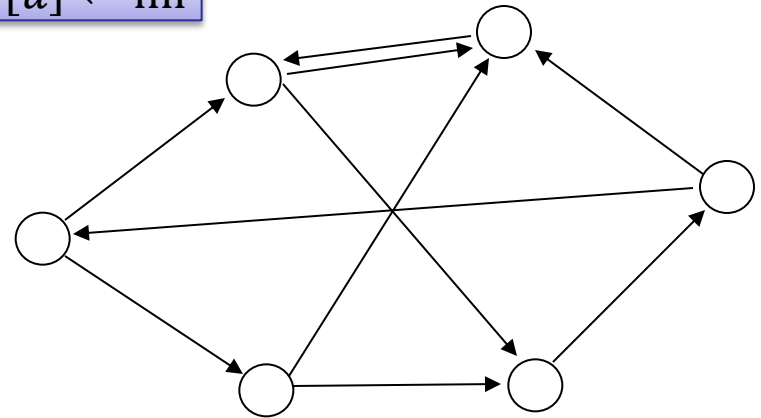
## Graphalgorithmen

DFS( $G$ )

1. **for each** vertex  $u \in V$  **do** color[ $u$ ]  $\leftarrow$  weiß;  $p[u] \leftarrow$  nil
2. time  $\leftarrow$  0
3. **for each** vertex  $u \in V$  **do**
4.   **if** color[ $u$ ] = weiß **then** DFS-Visit( $u$ )

DFS-Visit( $u$ )

1. color[ $u$ ]  $\leftarrow$  grau
2. time  $\leftarrow$  time + 1;  $d[u] \leftarrow$  time
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.   **if** color[ $v$ ] = weiß **then**  $p[v] \leftarrow u$  ; DFS-Visit( $v$ )
5. color[ $u$ ]  $\leftarrow$  schwarz
6. time  $\leftarrow$  time + 1;  $f[u] \leftarrow$  time



## Graphalgorithmen

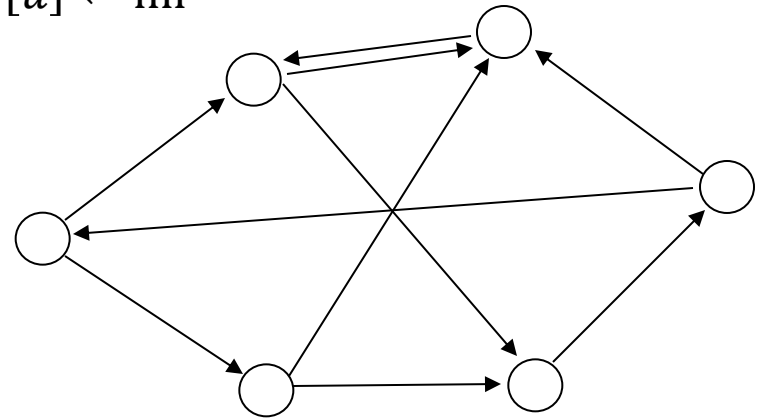
time = 0

DFS( $G$ )

1. **for each** vertex  $u \in V$  **do** color[ $u$ ]  $\leftarrow$  weiß;  $p[u] \leftarrow$  nil
2. time  $\leftarrow$  0
3. **for each** vertex  $u \in V$  **do**
4.   **if** color[ $u$ ] = weiß **then** DFS-Visit( $u$ )

DFS-Visit( $u$ )

1. color[ $u$ ]  $\leftarrow$  grau
2. time  $\leftarrow$  time + 1;  $d[u] \leftarrow$  time
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.   **if** color[ $v$ ] = weiß **then**  $p[v] \leftarrow u$  ; DFS-Visit( $v$ )
5. color[ $u$ ]  $\leftarrow$  schwarz
6. time  $\leftarrow$  time + 1;  $f[u] \leftarrow$  time



## Graphalgorithmen

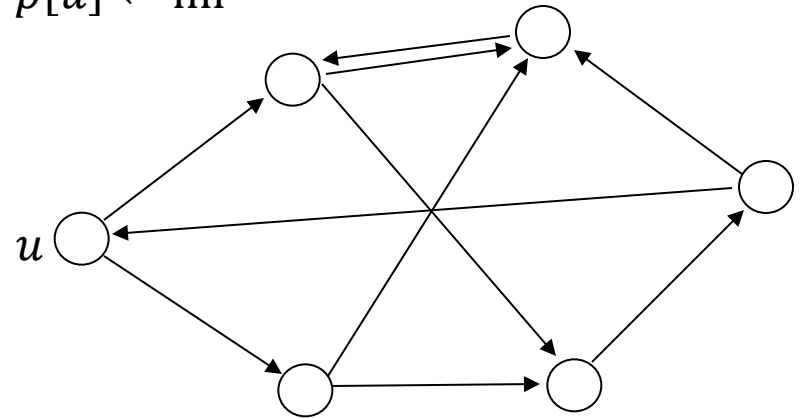
time = 0

DFS( $G$ )

1. **for each** vertex  $u \in V$  **do** color[ $u$ ]  $\leftarrow$  weiß;  $p[u] \leftarrow$  nil
2. time  $\leftarrow$  0
3. **for each** vertex  $u \in V$  **do**
4.   **if** color[ $u$ ] = weiß **then** DFS-Visit( $u$ )

DFS-Visit( $u$ )

1. color[ $u$ ]  $\leftarrow$  grau
2. time  $\leftarrow$  time + 1;  $d[u] \leftarrow$  time
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.   **if** color[ $v$ ] = weiß **then**  $p[v] \leftarrow u$  ; DFS-Visit( $v$ )
5. color[ $u$ ]  $\leftarrow$  schwarz
6. time  $\leftarrow$  time + 1;  $f[u] \leftarrow$  time





## Graphalgorithmen

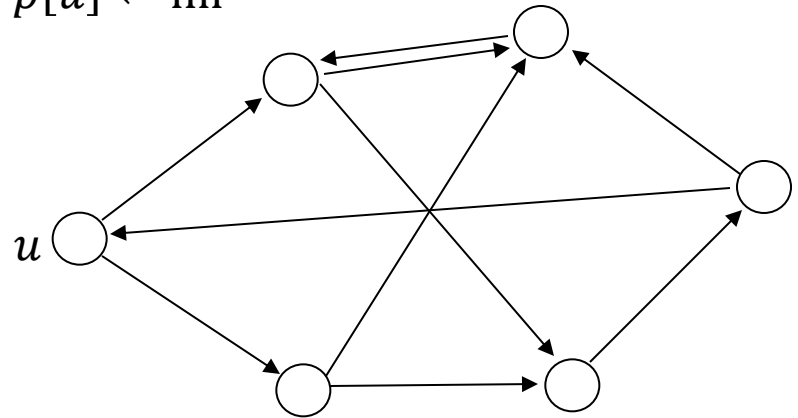
time = 0

DFS( $G$ )

1. **for each** vertex  $u \in V$  **do** color[ $u$ ]  $\leftarrow$  weiß;  $p[u] \leftarrow$  nil
2. time  $\leftarrow$  0
3. **for each** vertex  $u \in V$  **do**
4. **if** color[ $u$ ] = weiß **then** DFS-Visit( $u$ )

DFS-Visit( $u$ )

1. color[ $u$ ]  $\leftarrow$  grau
2. time  $\leftarrow$  time + 1;  $d[u] \leftarrow$  time
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.     **if** color[ $v$ ] = weiß **then**  $p[v] \leftarrow u$  ; DFS-Visit( $v$ )
5. color[ $u$ ]  $\leftarrow$  schwarz
6. time  $\leftarrow$  time + 1;  $f[u] \leftarrow$  time



## Graphalgorithmen

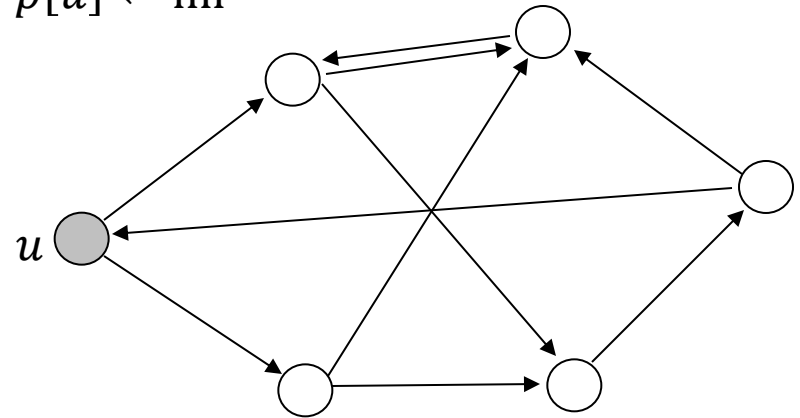
time = 0

DFS( $G$ )

1. **for each** vertex  $u \in V$  **do** color[ $u$ ]  $\leftarrow$  weiß;  $p[u] \leftarrow$  nil
2. time  $\leftarrow$  0
3. **for each** vertex  $u \in V$  **do**
4.   **if** color[ $u$ ] = weiß **then** DFS-Visit( $u$ )

DFS-Visit( $u$ )

1. color[ $u$ ]  $\leftarrow$  grau
2. time  $\leftarrow$  time + 1;  $d[u] \leftarrow$  time
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.   **if** color[ $v$ ] = weiß **then**  $p[v] \leftarrow u$  ; DFS-Visit( $v$ )
5. color[ $u$ ]  $\leftarrow$  schwarz
6. time  $\leftarrow$  time + 1;  $f[u] \leftarrow$  time



## Graphalgorithmen

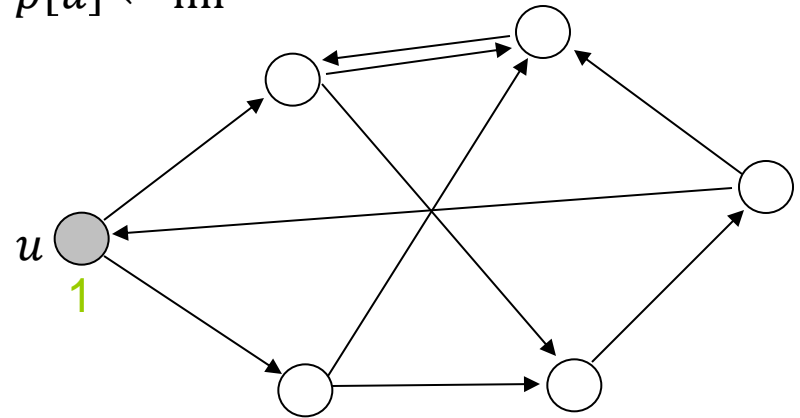
time = 1

DFS( $G$ )

1. **for each** vertex  $u \in V$  **do** color[ $u$ ]  $\leftarrow$  weiß;  $p[u] \leftarrow$  nil
2. time  $\leftarrow$  0
3. **for each** vertex  $u \in V$  **do**
4.   **if** color[ $u$ ] = weiß **then** DFS-Visit( $u$ )

DFS-Visit( $u$ )

1. color[ $u$ ]  $\leftarrow$  grau
2. time  $\leftarrow$  time + 1;  $d[u] \leftarrow$  time
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.   **if** color[ $v$ ] = weiß **then**  $p[v] \leftarrow u$  ; DFS-Visit( $v$ )
5. color[ $u$ ]  $\leftarrow$  schwarz
6. time  $\leftarrow$  time + 1;  $f[u] \leftarrow$  time



## Graphalgorithmen

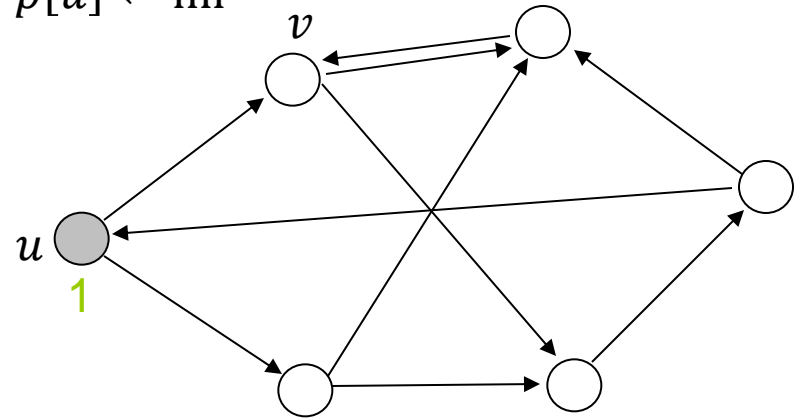
time = 1

DFS( $G$ )

1. **for each** vertex  $u \in V$  **do** color[ $u$ ]  $\leftarrow$  weiß;  $p[u] \leftarrow$  nil
2. time  $\leftarrow$  0
3. **for each** vertex  $u \in V$  **do**
4.   **if** color[ $u$ ] = weiß **then** DFS-Visit( $u$ )

DFS-Visit( $u$ )

1. color[ $u$ ]  $\leftarrow$  grau
2. time  $\leftarrow$  time + 1;  $d[u] \leftarrow$  time
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.   **if** color[ $v$ ] = weiß **then**  $p[v] \leftarrow u$  ; DFS-Visit( $v$ )
5. color[ $u$ ]  $\leftarrow$  schwarz
6. time  $\leftarrow$  time + 1;  $f[u] \leftarrow$  time



## Graphalgorithmen

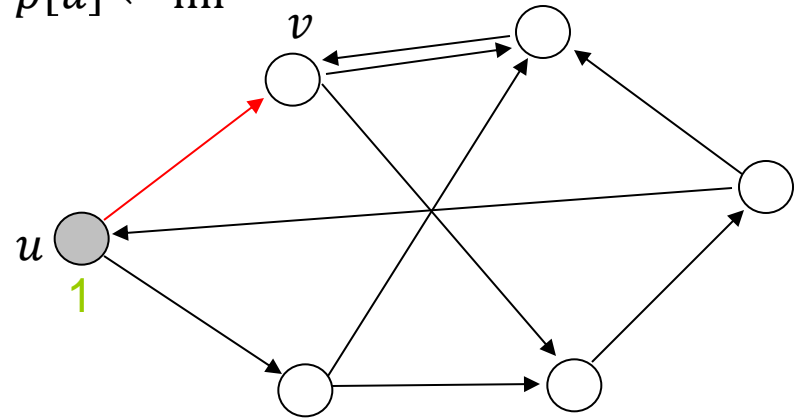
time = 1

DFS( $G$ )

1. **for each** vertex  $u \in V$  **do** color[ $u$ ]  $\leftarrow$  weiß;  $p[u] \leftarrow$  nil
2. time  $\leftarrow$  0
3. **for each** vertex  $u \in V$  **do**
4.   **if** color[ $u$ ] = weiß **then** DFS-Visit( $u$ )

DFS-Visit( $u$ )

1. color[ $u$ ]  $\leftarrow$  grau
2. time  $\leftarrow$  time + 1;  $d[u] \leftarrow$  time
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.   **if** color[ $v$ ] = weiß **then**  $p[v] \leftarrow u$  ; DFS-Visit( $v$ )
5. color[ $u$ ]  $\leftarrow$  schwarz
6. time  $\leftarrow$  time + 1;  $f[u] \leftarrow$  time



## Graphalgorithmen

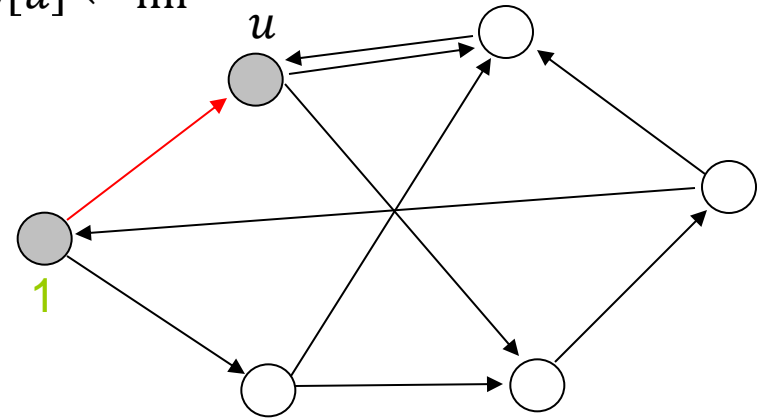
DFS( $G$ )

1. **for each** vertex  $u \in V$  **do** color[ $u$ ]  $\leftarrow$  weiß;  $p[u] \leftarrow$  nil
2. time  $\leftarrow$  0
3. **for each** vertex  $u \in V$  **do**
4.   **if** color[ $u$ ] = weiß **then** DFS-Visit( $u$ )

DFS-Visit( $u$ )

1. color[ $u$ ]  $\leftarrow$  grau
2. time  $\leftarrow$  time + 1;  $d[u] \leftarrow$  time
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.   **if** color[ $v$ ] = weiß **then**  $p[v] \leftarrow u$  ; DFS-Visit( $v$ )
5. color[ $u$ ]  $\leftarrow$  schwarz
6. time  $\leftarrow$  time + 1;  $f[u] \leftarrow$  time

time = 1



## Graphalgorithmen

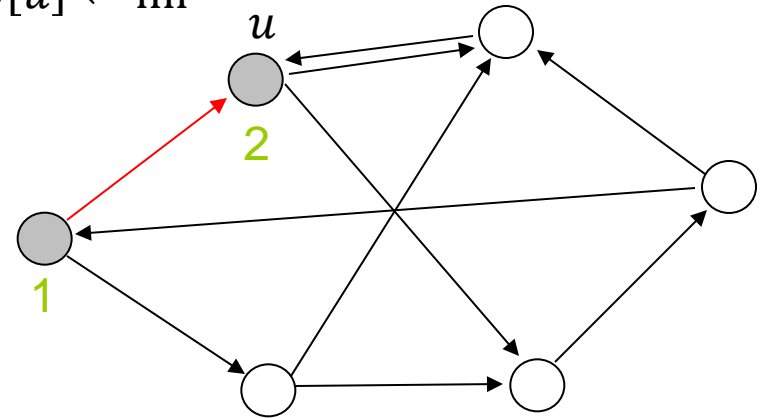
DFS( $G$ )

1. **for each** vertex  $u \in V$  **do** color[ $u$ ]  $\leftarrow$  weiß;  $p[u] \leftarrow$  nil
2. time  $\leftarrow$  0
3. **for each** vertex  $u \in V$  **do**
4.   **if** color[ $u$ ] = weiß **then** DFS-Visit( $u$ )

DFS-Visit( $u$ )

1. color[ $u$ ]  $\leftarrow$  grau
2. time  $\leftarrow$  time + 1;  $d[u] \leftarrow$  time
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.   **if** color[ $v$ ] = weiß **then**  $p[v] \leftarrow u$  ; DFS-Visit( $v$ )
5. color[ $u$ ]  $\leftarrow$  schwarz
6. time  $\leftarrow$  time + 1;  $f[u] \leftarrow$  time

time = 2



## Graphalgorithmen

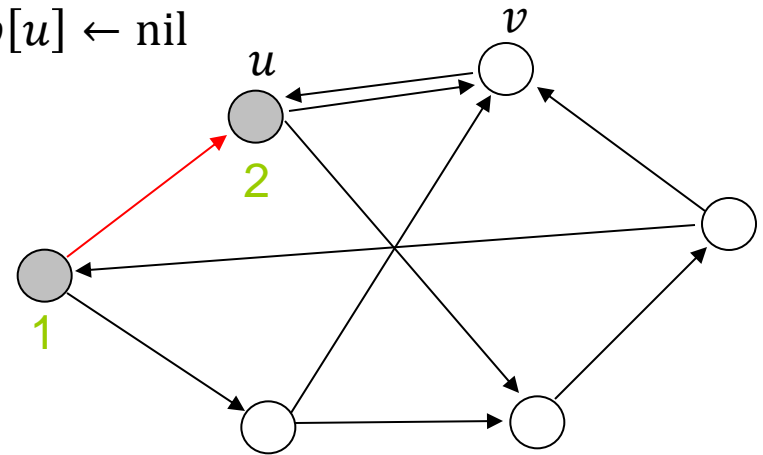
DFS( $G$ )

1. **for each** vertex  $u \in V$  **do** color[ $u$ ]  $\leftarrow$  weiß;  $p[u] \leftarrow$  nil
2. time  $\leftarrow$  0
3. **for each** vertex  $u \in V$  **do**
4.   **if** color[ $u$ ] = weiß **then** DFS-Visit( $u$ )

DFS-Visit( $u$ )

1. color[ $u$ ]  $\leftarrow$  grau
2. time  $\leftarrow$  time + 1;  $d[u] \leftarrow$  time
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.   **if** color[ $v$ ] = weiß **then**  $p[v] \leftarrow u$  ; DFS-Visit( $v$ )
5. color[ $u$ ]  $\leftarrow$  schwarz
6. time  $\leftarrow$  time + 1;  $f[u] \leftarrow$  time

time = 2





## Graphalgorithmen

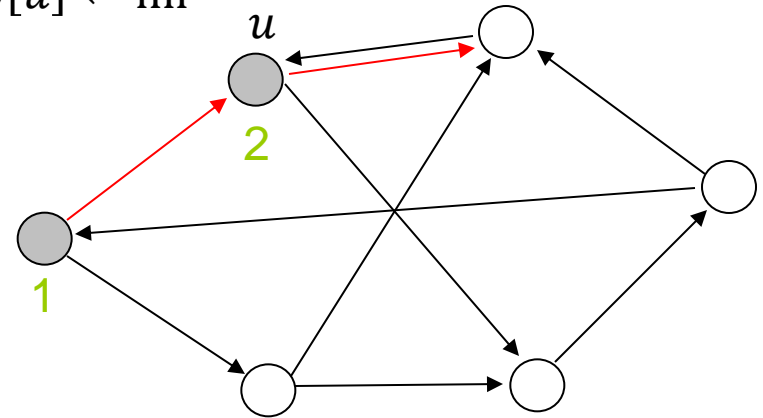
time = 2

DFS( $G$ )

1. **for each** vertex  $u \in V$  **do** color[ $u$ ]  $\leftarrow$  weiß;  $p[u] \leftarrow$  nil
2. time  $\leftarrow$  0
3. **for each** vertex  $u \in V$  **do**
4.   **if** color[ $u$ ] = weiß **then** DFS-Visit( $u$ )

DFS-Visit( $u$ )

1. color[ $u$ ]  $\leftarrow$  grau
2. time  $\leftarrow$  time + 1;  $d[u] \leftarrow$  time
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.   **if** color[ $v$ ] = weiß **then**  $p[v] \leftarrow u$  ; DFS-Visit( $v$ )
5. color[ $u$ ]  $\leftarrow$  schwarz
6. time  $\leftarrow$  time + 1;  $f[u] \leftarrow$  time



## Graphalgorithmen

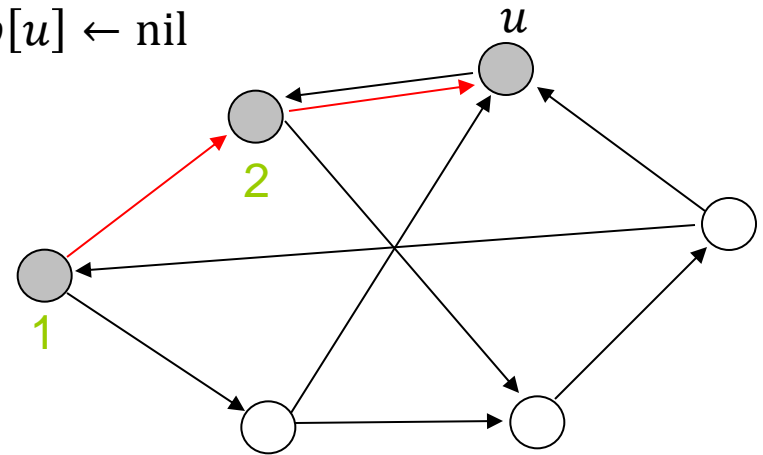
DFS( $G$ )

1. **for each** vertex  $u \in V$  **do** color[ $u$ ]  $\leftarrow$  weiß;  $p[u] \leftarrow$  nil
2. time  $\leftarrow$  0
3. **for each** vertex  $u \in V$  **do**
4.   **if** color[ $u$ ] = weiß **then** DFS-Visit( $u$ )

DFS-Visit( $u$ )

1. color[ $u$ ]  $\leftarrow$  grau
2. time  $\leftarrow$  time + 1;  $d[u] \leftarrow$  time
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.   **if** color[ $v$ ] = weiß **then**  $p[v] \leftarrow u$  ; DFS-Visit( $v$ )
5. color[ $u$ ]  $\leftarrow$  schwarz
6. time  $\leftarrow$  time + 1;  $f[u] \leftarrow$  time

time = 2



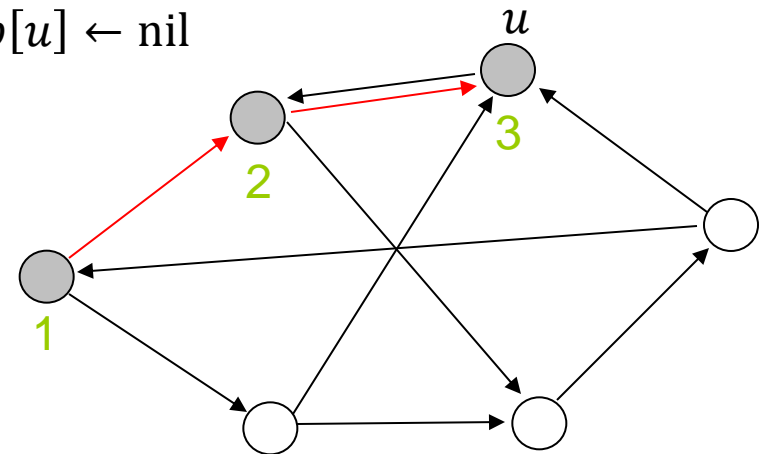
## Graphalgorithmen

DFS( $G$ )

1. **for each** vertex  $u \in V$  **do** color[ $u$ ]  $\leftarrow$  weiß;  $p[u] \leftarrow$  nil
2. time  $\leftarrow$  0
3. **for each** vertex  $u \in V$  **do**
4.   **if** color[ $u$ ] = weiß **then** DFS-Visit( $u$ )

DFS-Visit( $u$ )

1. color[ $u$ ]  $\leftarrow$  grau
2. time  $\leftarrow$  time + 1;  $d[u] \leftarrow$  time
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.   **if** color[ $v$ ] = weiß **then**  $p[v] \leftarrow u$  ; DFS-Visit( $v$ )
5. color[ $u$ ]  $\leftarrow$  schwarz
6. time  $\leftarrow$  time + 1;  $f[u] \leftarrow$  time



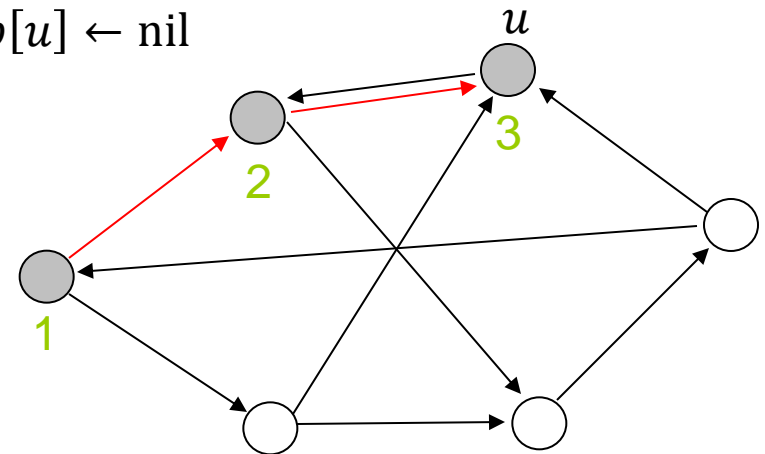
## Graphalgorithmen

DFS( $G$ )

1. **for each** vertex  $u \in V$  **do** color[ $u$ ]  $\leftarrow$  weiß;  $p[u] \leftarrow$  nil
2. time  $\leftarrow$  0
3. **for each** vertex  $u \in V$  **do**
4.   **if** color[ $u$ ] = weiß **then** DFS-Visit( $u$ )

DFS-Visit( $u$ )

1. color[ $u$ ]  $\leftarrow$  grau
2. time  $\leftarrow$  time + 1;  $d[u] \leftarrow$  time
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.   **if** color[ $v$ ] = weiß **then**  $p[v] \leftarrow u$  ; DFS-Visit( $v$ )
5. color[ $u$ ]  $\leftarrow$  schwarz
6. time  $\leftarrow$  time + 1;  $f[u] \leftarrow$  time



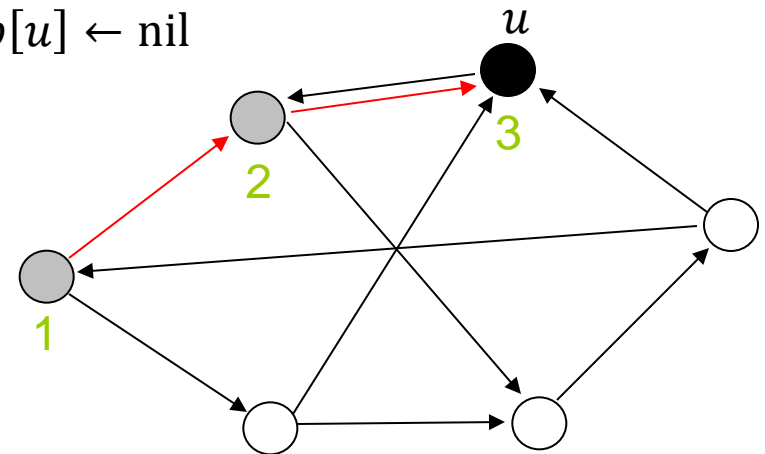
## Graphalgorithmen

DFS( $G$ )

1. **for each** vertex  $u \in V$  **do**  $\text{color}[u] \leftarrow \text{weiß}$ ;  $p[u] \leftarrow \text{nil}$
2.  $\text{time} \leftarrow 0$
3. **for each** vertex  $u \in V$  **do**
4.   **if**  $\text{color}[u] = \text{weiß}$  **then** DFS-Visit( $u$ )

DFS-Visit( $u$ )

1.  $\text{color}[u] \leftarrow \text{grau}$
2.  $\text{time} \leftarrow \text{time} + 1$ ;  $d[u] \leftarrow \text{time}$
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.   **if**  $\text{color}[v] = \text{weiß}$  **then**  $p[v] \leftarrow u$ ; DFS-Visit( $v$ )
5.  $\text{color}[u] \leftarrow \text{schwarz}$
6.  $\text{time} \leftarrow \text{time} + 1$ ;  $f[u] \leftarrow \text{time}$



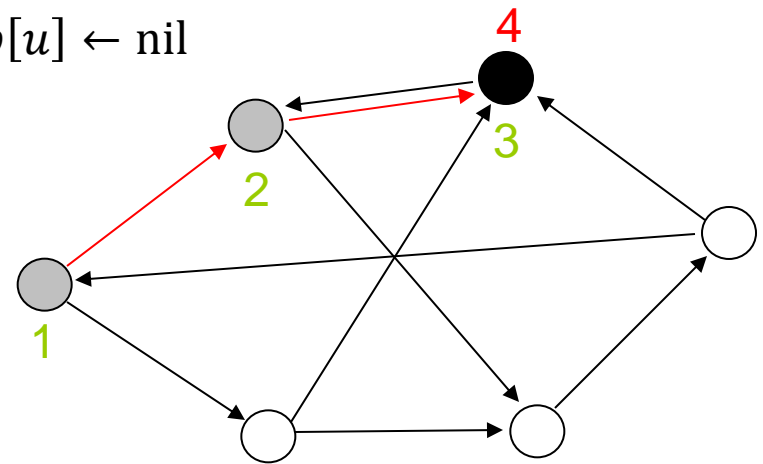
# Graphalgorithmen

$$\text{DFS}(G)$$

1. **for each** vertex  $u \in V$  **do** color[ $u$ ]  $\leftarrow$  weiß;  $p[u] \leftarrow$  nil
2. time  $\leftarrow$  0
3. **for each** vertex  $u \in V$  **do**
4.     **if** color[ $u$ ] = weiß **then** DFS-Visit( $u$ )

$$\text{DFS-Visit}(u)$$

1.  $\text{color}[u] \leftarrow \text{grau}$
2.  $\text{time} \leftarrow \text{time} + 1; d[u] \leftarrow \text{time}$
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.     **if**  $\text{color}[v] = \text{weiß}$  **then**  $p[v] \leftarrow u; \text{DFS-Visit}(v)$
5.  $\text{color}[u] \leftarrow \text{schwarz}$
6.  $\text{time} \leftarrow \text{time} + 1; f[u] \leftarrow \text{time}$



time = 4

## Graphalgorithmen

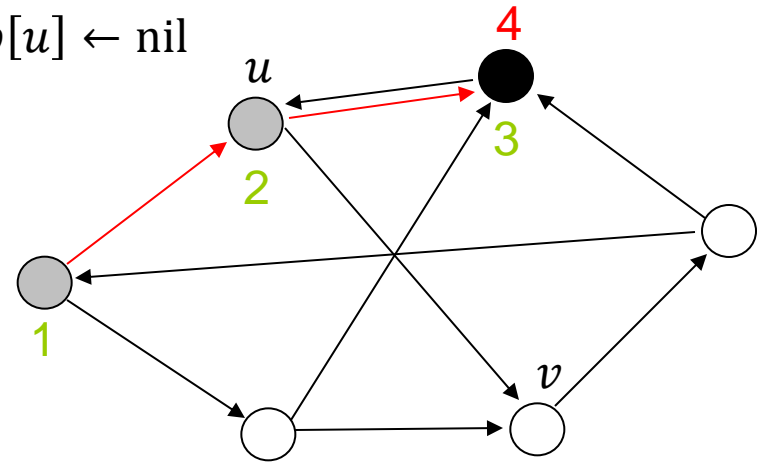
DFS( $G$ )

1. **for each** vertex  $u \in V$  **do** color[ $u$ ]  $\leftarrow$  weiß;  $p[u] \leftarrow$  nil
2. time  $\leftarrow$  0
3. **for each** vertex  $u \in V$  **do**
4.   **if** color[ $u$ ] = weiß **then** DFS-Visit( $u$ )

DFS-Visit( $u$ )

1. color[ $u$ ]  $\leftarrow$  grau
2. time  $\leftarrow$  time + 1;  $d[u] \leftarrow$  time
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.   **if** color[ $v$ ] = weiß **then**  $p[v] \leftarrow u$  ; DFS-Visit( $v$ )
5. color[ $u$ ]  $\leftarrow$  schwarz
6. time  $\leftarrow$  time + 1;  $f[u] \leftarrow$  time

time = 4



## Graphalgorithmen

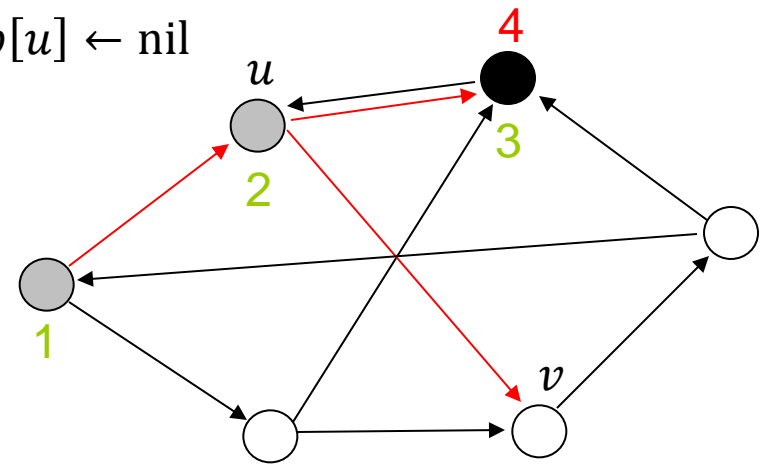
DFS( $G$ )

1. **for each** vertex  $u \in V$  **do** color[ $u$ ]  $\leftarrow$  weiß;  $p[u] \leftarrow$  nil
2. time  $\leftarrow$  0
3. **for each** vertex  $u \in V$  **do**
4.   **if** color[ $u$ ] = weiß **then** DFS-Visit( $u$ )

DFS-Visit( $u$ )

1. color[ $u$ ]  $\leftarrow$  grau
2. time  $\leftarrow$  time + 1;  $d[u] \leftarrow$  time
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.   **if** color[ $v$ ] = weiß **then**  $p[v] \leftarrow u$  ; DFS-Visit( $v$ )
5. color[ $u$ ]  $\leftarrow$  schwarz
6. time  $\leftarrow$  time + 1;  $f[u] \leftarrow$  time

time = 4





## Graphalgorithmen

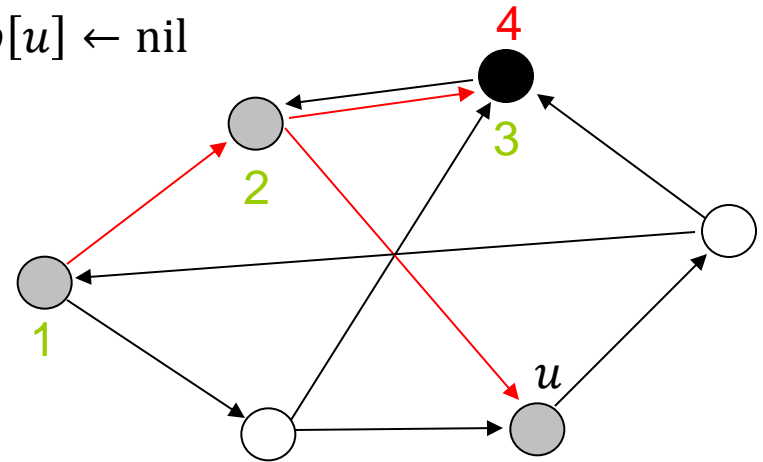
DFS( $G$ )

1. **for each** vertex  $u \in V$  **do** color[ $u$ ]  $\leftarrow$  weiß;  $p[u] \leftarrow$  nil
2. time  $\leftarrow$  0
3. **for each** vertex  $u \in V$  **do**
4.   **if** color[ $u$ ] = weiß **then** DFS-Visit( $u$ )

DFS-Visit( $u$ )

1. color[ $u$ ]  $\leftarrow$  grau
2. time  $\leftarrow$  time + 1;  $d[u] \leftarrow$  time
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.   **if** color[ $v$ ] = weiß **then**  $p[v] \leftarrow u$  ; DFS-Visit( $v$ )
5. color[ $u$ ]  $\leftarrow$  schwarz
6. time  $\leftarrow$  time + 1;  $f[u] \leftarrow$  time

time = 4



## Graphalgorithmen

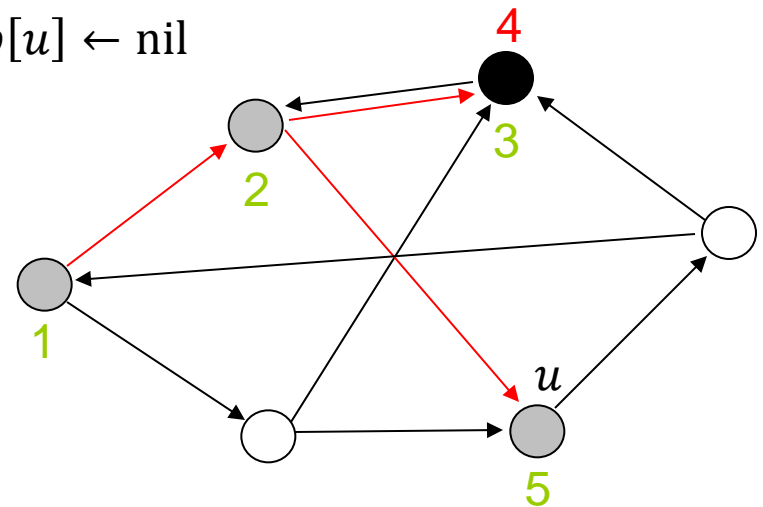
DFS( $G$ )

1. **for each** vertex  $u \in V$  **do** color[ $u$ ]  $\leftarrow$  weiß;  $p[u] \leftarrow$  nil
2. time  $\leftarrow$  0
3. **for each** vertex  $u \in V$  **do**
4.   **if** color[ $u$ ] = weiß **then** DFS-Visit( $u$ )

DFS-Visit( $u$ )

1. color[ $u$ ]  $\leftarrow$  grau
2. time  $\leftarrow$  time + 1;  $d[u] \leftarrow$  time
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.   **if** color[ $v$ ] = weiß **then**  $p[v] \leftarrow u$  ; DFS-Visit( $v$ )
5. color[ $u$ ]  $\leftarrow$  schwarz
6. time  $\leftarrow$  time + 1;  $f[u] \leftarrow$  time

time = 5



## Graphalgorithmen

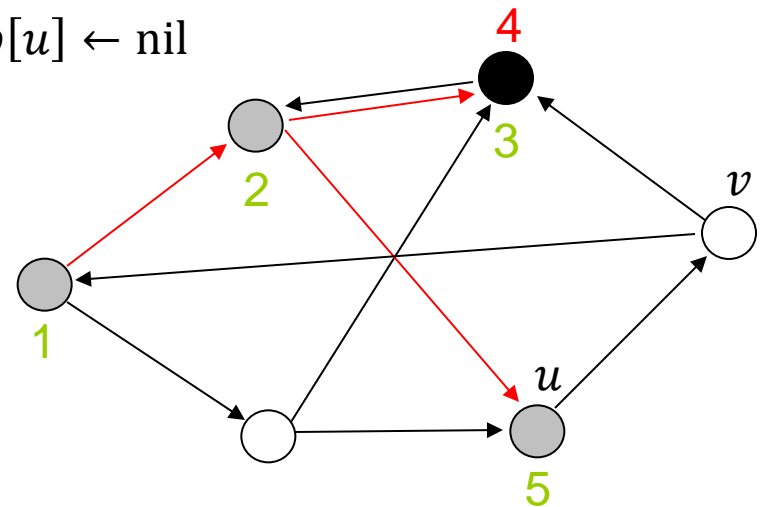
DFS( $G$ )

1. **for each** vertex  $u \in V$  **do** color[ $u$ ]  $\leftarrow$  weiß;  $p[u] \leftarrow$  nil
2. time  $\leftarrow$  0
3. **for each** vertex  $u \in V$  **do**
4.   **if** color[ $u$ ] = weiß **then** DFS-Visit( $u$ )

DFS-Visit( $u$ )

1. color[ $u$ ]  $\leftarrow$  grau
2. time  $\leftarrow$  time + 1;  $d[u] \leftarrow$  time
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.   **if** color[ $v$ ] = weiß **then**  $p[v] \leftarrow u$  ; DFS-Visit( $v$ )
5. color[ $u$ ]  $\leftarrow$  schwarz
6. time  $\leftarrow$  time + 1;  $f[u] \leftarrow$  time

time = 5



## Graphalgorithmen

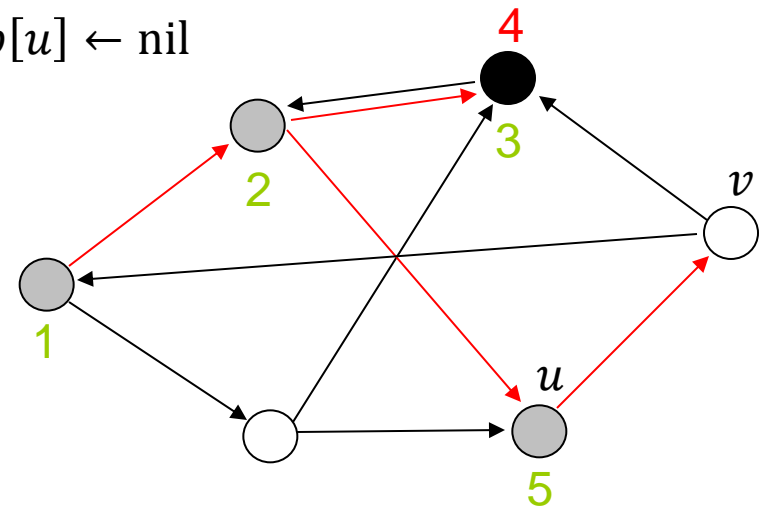
DFS( $G$ )

1. **for each** vertex  $u \in V$  **do** color[ $u$ ]  $\leftarrow$  weiß;  $p[u] \leftarrow$  nil
2. time  $\leftarrow$  0
3. **for each** vertex  $u \in V$  **do**
4.   **if** color[ $u$ ] = weiß **then** DFS-Visit( $u$ )

DFS-Visit( $u$ )

1. color[ $u$ ]  $\leftarrow$  grau
2. time  $\leftarrow$  time + 1;  $d[u] \leftarrow$  time
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.   **if** color[ $v$ ] = weiß **then**  $p[v] \leftarrow u$  ; DFS-Visit( $v$ )
5. color[ $u$ ]  $\leftarrow$  schwarz
6. time  $\leftarrow$  time + 1;  $f[u] \leftarrow$  time

time = 5



## Graphalgorithmen

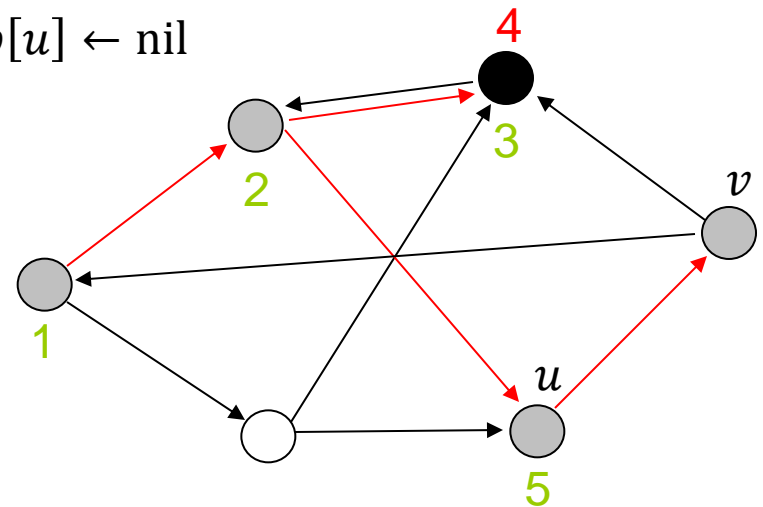
DFS( $G$ )

1. **for each** vertex  $u \in V$  **do** color[ $u$ ]  $\leftarrow$  weiß;  $p[u] \leftarrow$  nil
2. time  $\leftarrow$  0
3. **for each** vertex  $u \in V$  **do**
4.   **if** color[ $u$ ] = weiß **then** DFS-Visit( $u$ )

DFS-Visit( $u$ )

1. color[ $u$ ]  $\leftarrow$  grau
2. time  $\leftarrow$  time + 1;  $d[u] \leftarrow$  time
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.   **if** color[ $v$ ] = weiß **then**  $p[v] \leftarrow u$  ; DFS-Visit( $v$ )
5. color[ $u$ ]  $\leftarrow$  schwarz
6. time  $\leftarrow$  time + 1;  $f[u] \leftarrow$  time

time = 5



## Graphalgorithmen

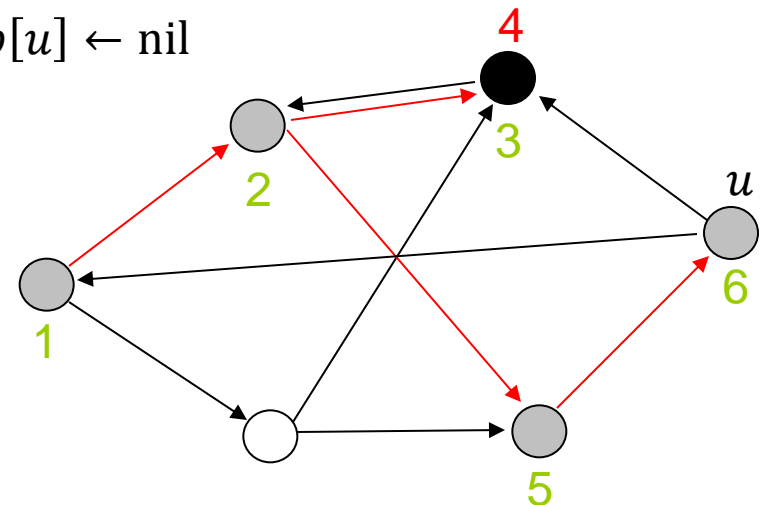
DFS( $G$ )

1. **for each** vertex  $u \in V$  **do** color[ $u$ ]  $\leftarrow$  weiß;  $p[u] \leftarrow$  nil
2. time  $\leftarrow$  0
3. **for each** vertex  $u \in V$  **do**
4.   **if** color[ $u$ ] = weiß **then** DFS-Visit( $u$ )

DFS-Visit( $u$ )

1. color[ $u$ ]  $\leftarrow$  grau
2. time  $\leftarrow$  time + 1;  $d[u] \leftarrow$  time
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.   **if** color[ $v$ ] = weiß **then**  $p[v] \leftarrow u$  ; DFS-Visit( $v$ )
5. color[ $u$ ]  $\leftarrow$  schwarz
6. time  $\leftarrow$  time + 1;  $f[u] \leftarrow$  time

time = 6



## Graphalgorithmen

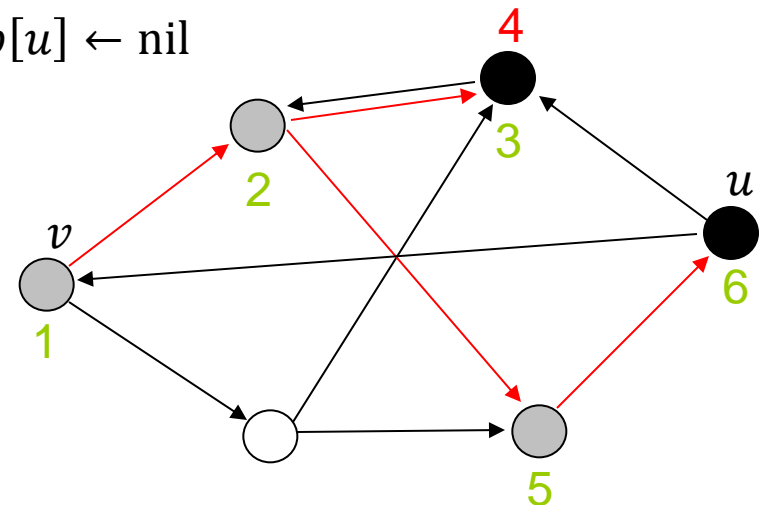
DFS( $G$ )

1. **for each** vertex  $u \in V$  **do** color[ $u$ ]  $\leftarrow$  weiß;  $p[u] \leftarrow$  nil
2. time  $\leftarrow$  0
3. **for each** vertex  $u \in V$  **do**
4.   **if** color[ $u$ ] = weiß **then** DFS-Visit( $u$ )

DFS-Visit( $u$ )

1. color[ $u$ ]  $\leftarrow$  grau
2. time  $\leftarrow$  time + 1;  $d[u] \leftarrow$  time
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.   **if** color[ $v$ ] = weiß **then**  $p[v] \leftarrow u$  ; DFS-Visit( $v$ )
5. color[ $u$ ]  $\leftarrow$  schwarz
6. time  $\leftarrow$  time + 1;  $f[u] \leftarrow$  time

time = 6



## Graphalgorithmen

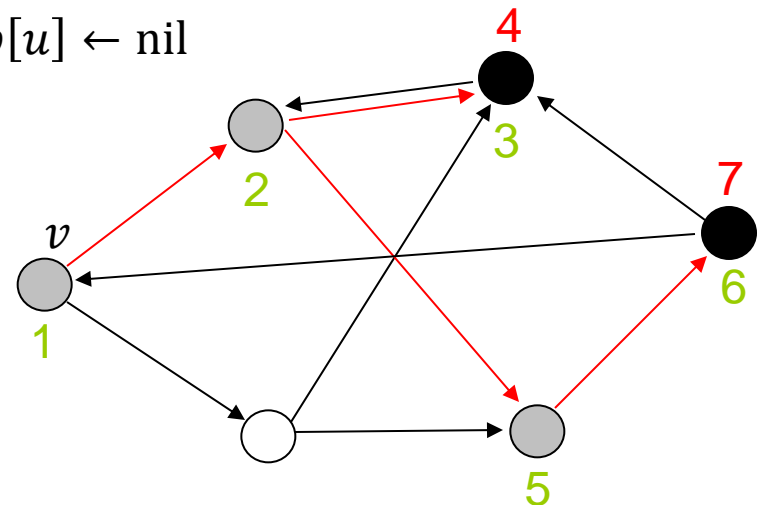
DFS( $G$ )

1. **for each** vertex  $u \in V$  **do** color[ $u$ ]  $\leftarrow$  weiß;  $p[u] \leftarrow$  nil
2. time  $\leftarrow$  0
3. **for each** vertex  $u \in V$  **do**
4.   **if** color[ $u$ ] = weiß **then** DFS-Visit( $u$ )

DFS-Visit( $u$ )

1. color[ $u$ ]  $\leftarrow$  grau
2. time  $\leftarrow$  time + 1;  $d[u] \leftarrow$  time
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.   **if** color[ $v$ ] = weiß **then**  $p[v] \leftarrow u$  ; DFS-Visit( $v$ )
5. color[ $u$ ]  $\leftarrow$  schwarz
6. time  $\leftarrow$  time + 1;  $f[u] \leftarrow$  time

time = 7





## Graphalgorithmen

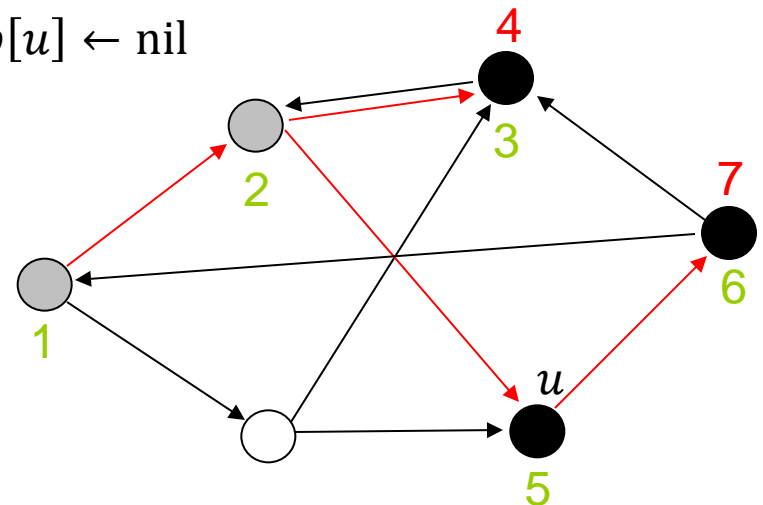
DFS( $G$ )

1. **for each** vertex  $u \in V$  **do** color[ $u$ ]  $\leftarrow$  weiß;  $p[u] \leftarrow$  nil
2. time  $\leftarrow$  0
3. **for each** vertex  $u \in V$  **do**
4.   **if** color[ $u$ ] = weiß **then** DFS-Visit( $u$ )

DFS-Visit( $u$ )

1. color[ $u$ ]  $\leftarrow$  grau
2. time  $\leftarrow$  time + 1;  $d[u] \leftarrow$  time
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.   **if** color[ $v$ ] = weiß **then**  $p[v] \leftarrow u$  ; DFS-Visit( $v$ )
5. color[ $u$ ]  $\leftarrow$  schwarz
6. time  $\leftarrow$  time + 1;  $f[u] \leftarrow$  time

time = 7



## Graphalgorithmen

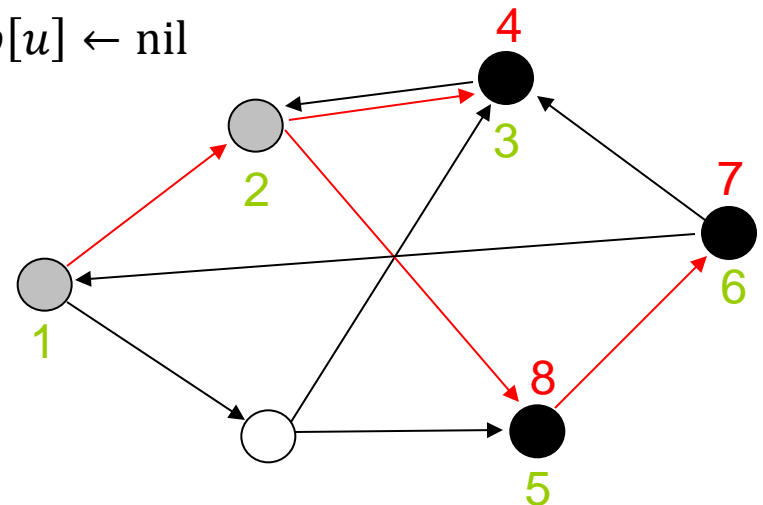
DFS( $G$ )

1. **for each** vertex  $u \in V$  **do** color[ $u$ ]  $\leftarrow$  weiß;  $p[u] \leftarrow$  nil
2. time  $\leftarrow$  0
3. **for each** vertex  $u \in V$  **do**
4.   **if** color[ $u$ ] = weiß **then** DFS-Visit( $u$ )

DFS-Visit( $u$ )

1. color[ $u$ ]  $\leftarrow$  grau
2. time  $\leftarrow$  time + 1;  $d[u] \leftarrow$  time
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.   **if** color[ $v$ ] = weiß **then**  $p[v] \leftarrow u$  ; DFS-Visit( $v$ )
5. color[ $u$ ]  $\leftarrow$  schwarz
6. time  $\leftarrow$  time + 1;  $f[u] \leftarrow$  time

time = 8



## Graphalgorithmen

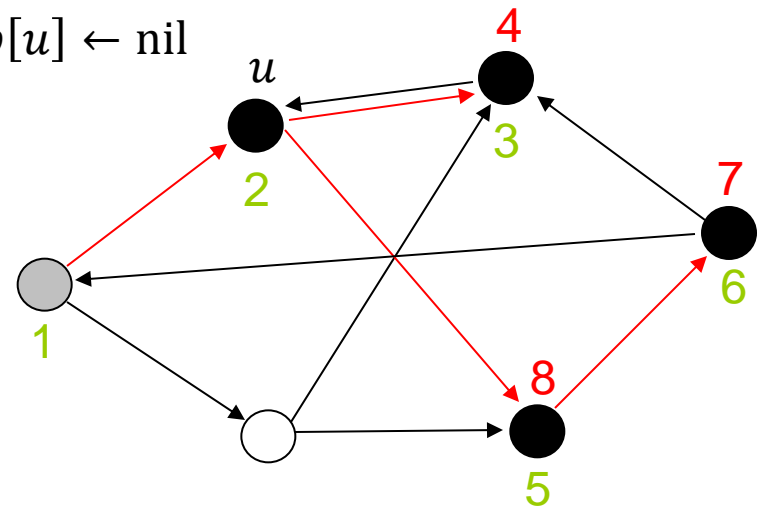
DFS( $G$ )

1. **for each** vertex  $u \in V$  **do** color[ $u$ ]  $\leftarrow$  weiß;  $p[u] \leftarrow$  nil
2. time  $\leftarrow$  0
3. **for each** vertex  $u \in V$  **do**
4.   **if** color[ $u$ ] = weiß **then** DFS-Visit( $u$ )

DFS-Visit( $u$ )

1. color[ $u$ ]  $\leftarrow$  grau
2. time  $\leftarrow$  time + 1;  $d[u] \leftarrow$  time
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.   **if** color[ $v$ ] = weiß **then**  $p[v] \leftarrow u$  ; DFS-Visit( $v$ )
5. color[ $u$ ]  $\leftarrow$  schwarz
6. time  $\leftarrow$  time + 1;  $f[u] \leftarrow$  time

time = 8



## Graphalgorithmen

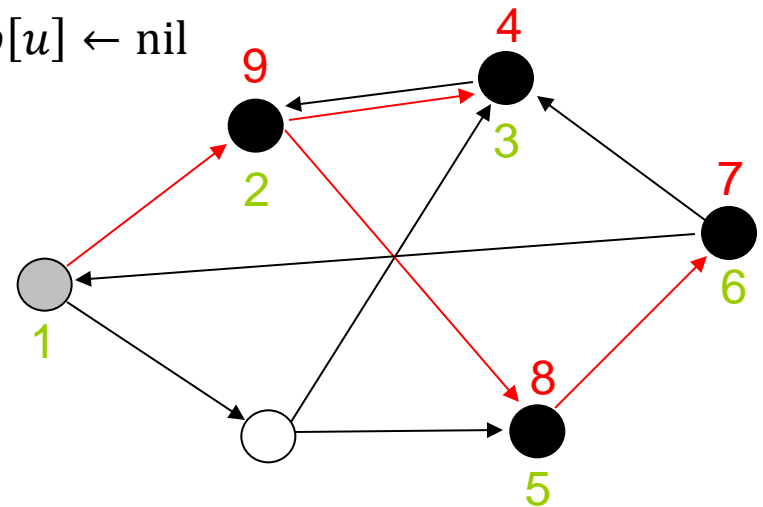
DFS( $G$ )

1. **for each** vertex  $u \in V$  **do** color[ $u$ ]  $\leftarrow$  weiß;  $p[u] \leftarrow$  nil
2. time  $\leftarrow$  0
3. **for each** vertex  $u \in V$  **do**
4.   **if** color[ $u$ ] = weiß **then** DFS-Visit( $u$ )

DFS-Visit( $u$ )

1. color[ $u$ ]  $\leftarrow$  grau
2. time  $\leftarrow$  time + 1;  $d[u] \leftarrow$  time
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.   **if** color[ $v$ ] = weiß **then**  $p[v] \leftarrow u$  ; DFS-Visit( $v$ )
5. color[ $u$ ]  $\leftarrow$  schwarz
6. time  $\leftarrow$  time + 1;  $f[u] \leftarrow$  time

time = 9



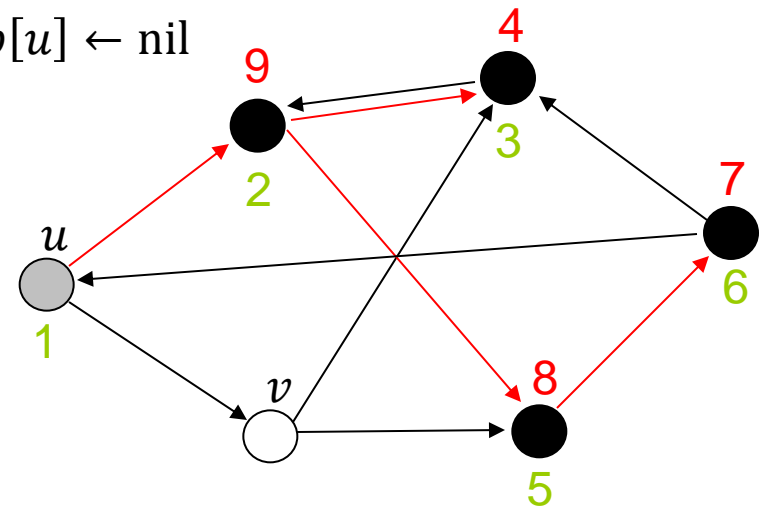
## Graphalgorithmen

DFS( $G$ )

1. **for each** vertex  $u \in V$  **do** color[ $u$ ]  $\leftarrow$  weiß;  $p[u] \leftarrow$  nil
2. time  $\leftarrow$  0
3. **for each** vertex  $u \in V$  **do**
4.   **if** color[ $u$ ] = weiß **then** DFS-Visit( $u$ )

DFS-Visit( $u$ )

1. color[ $u$ ]  $\leftarrow$  grau
2. time  $\leftarrow$  time + 1;  $d[u] \leftarrow$  time
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.   **if** color[ $v$ ] = weiß **then**  $p[v] \leftarrow u$  ; DFS-Visit( $v$ )
5. color[ $u$ ]  $\leftarrow$  schwarz
6. time  $\leftarrow$  time + 1;  $f[u] \leftarrow$  time



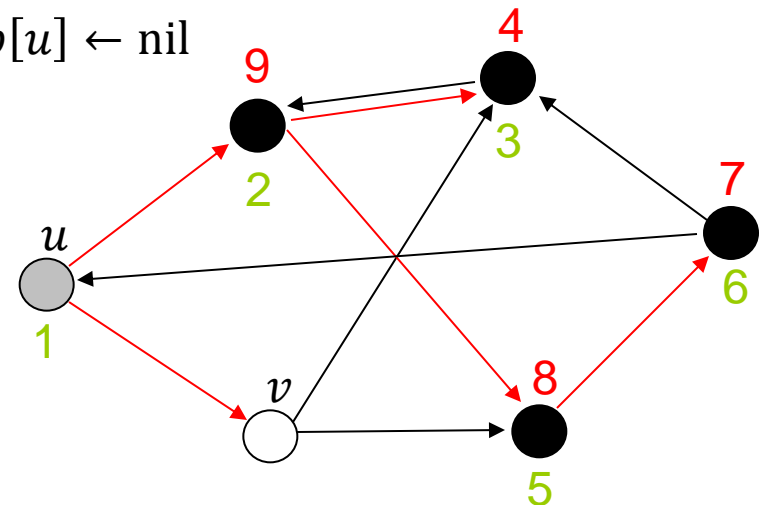
## Graphalgorithmen

DFS( $G$ )

1. **for each** vertex  $u \in V$  **do** color[ $u$ ]  $\leftarrow$  weiß;  $p[u] \leftarrow$  nil
2. time  $\leftarrow$  0
3. **for each** vertex  $u \in V$  **do**
4.   **if** color[ $u$ ] = weiß **then** DFS-Visit( $u$ )

DFS-Visit( $u$ )

1. color[ $u$ ]  $\leftarrow$  grau
2. time  $\leftarrow$  time + 1;  $d[u] \leftarrow$  time
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.   **if** color[ $v$ ] = weiß **then**  $p[v] \leftarrow u$  ; DFS-Visit( $v$ )
5. color[ $u$ ]  $\leftarrow$  schwarz
6. time  $\leftarrow$  time + 1;  $f[u] \leftarrow$  time



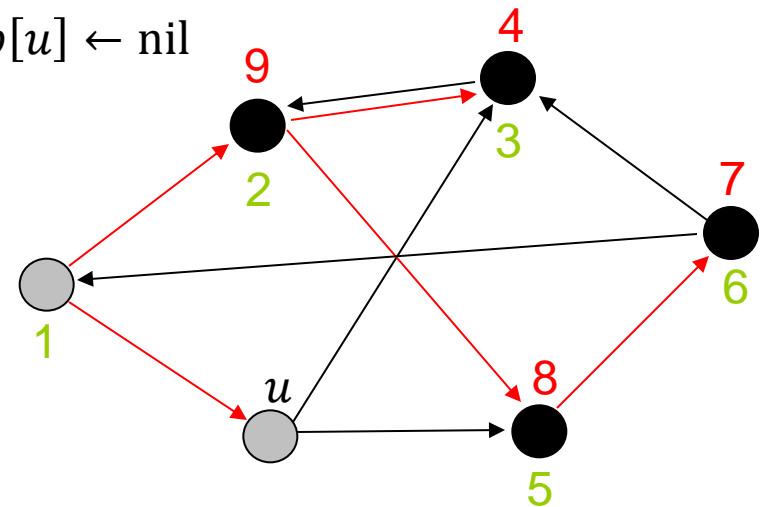
## Graphalgorithmen

DFS( $G$ )

1. **for each** vertex  $u \in V$  **do** color[ $u$ ]  $\leftarrow$  weiß;  $p[u] \leftarrow$  nil
2. time  $\leftarrow$  0
3. **for each** vertex  $u \in V$  **do**
4.   **if** color[ $u$ ] = weiß **then** DFS-Visit( $u$ )

DFS-Visit( $u$ )

1. color[ $u$ ]  $\leftarrow$  grau
2. time  $\leftarrow$  time + 1;  $d[u] \leftarrow$  time
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.   **if** color[ $v$ ] = weiß **then**  $p[v] \leftarrow u$  ; DFS-Visit( $v$ )
5. color[ $u$ ]  $\leftarrow$  schwarz
6. time  $\leftarrow$  time + 1;  $f[u] \leftarrow$  time



## Graphalgorithmen

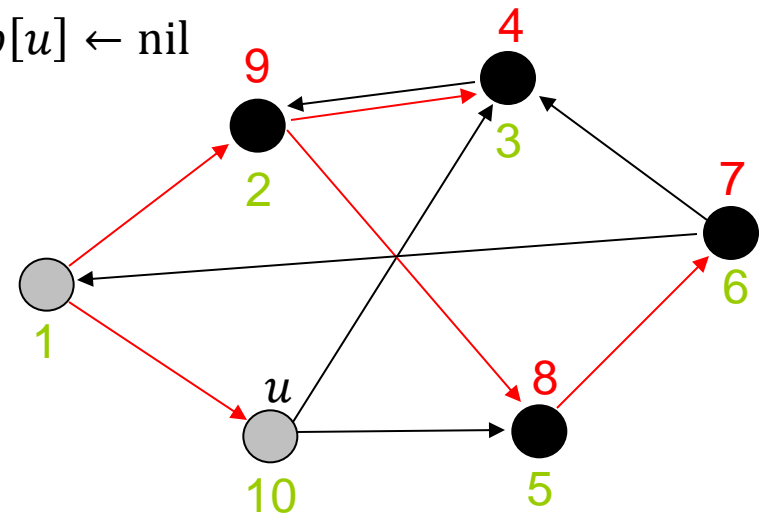
DFS( $G$ )

1. **for each** vertex  $u \in V$  **do** color[ $u$ ]  $\leftarrow$  weiß;  $p[u] \leftarrow$  nil
2. time  $\leftarrow$  0
3. **for each** vertex  $u \in V$  **do**
4.   **if** color[ $u$ ] = weiß **then** DFS-Visit( $u$ )

DFS-Visit( $u$ )

1. color[ $u$ ]  $\leftarrow$  grau
2. time  $\leftarrow$  time + 1;  $d[u] \leftarrow$  time
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.   **if** color[ $v$ ] = weiß **then**  $p[v] \leftarrow u$  ; DFS-Visit( $v$ )
5. color[ $u$ ]  $\leftarrow$  schwarz
6. time  $\leftarrow$  time + 1;  $f[u] \leftarrow$  time

time = 10





## Graphalgorithmen

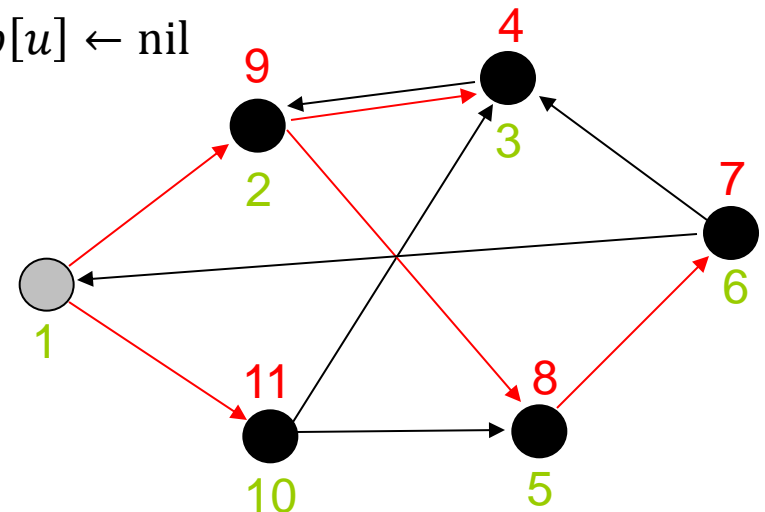
DFS( $G$ )

1. **for each** vertex  $u \in V$  **do** color[ $u$ ]  $\leftarrow$  weiß;  $p[u] \leftarrow$  nil
2. time  $\leftarrow$  0
3. **for each** vertex  $u \in V$  **do**
4.   **if** color[ $u$ ] = weiß **then** DFS-Visit( $u$ )

DFS-Visit( $u$ )

1. color[ $u$ ]  $\leftarrow$  grau
2. time  $\leftarrow$  time + 1;  $d[u] \leftarrow$  time
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.   **if** color[ $v$ ] = weiß **then**  $p[v] \leftarrow u$  ; DFS-Visit( $v$ )
5. color[ $u$ ]  $\leftarrow$  schwarz
6. time  $\leftarrow$  time + 1;  $f[u] \leftarrow$  time

time = 11



## Graphalgorithmen

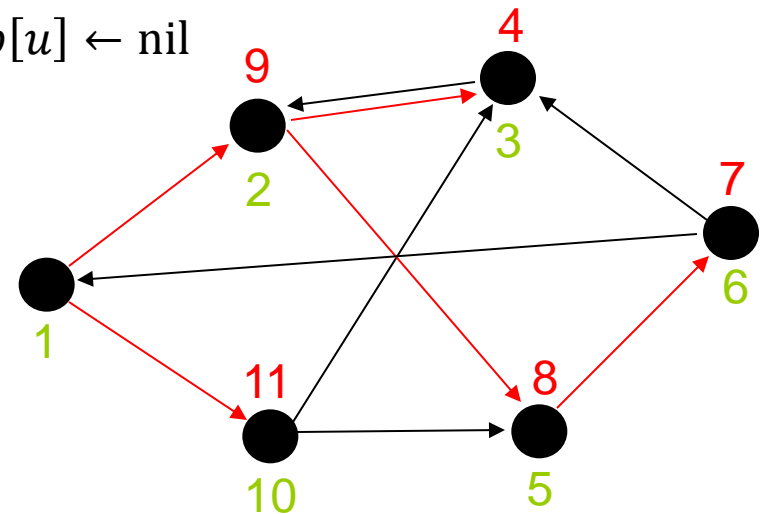
DFS( $G$ )

1. **for each** vertex  $u \in V$  **do** color[ $u$ ]  $\leftarrow$  weiß;  $p[u] \leftarrow$  nil
2. time  $\leftarrow$  0
3. **for each** vertex  $u \in V$  **do**
4.   **if** color[ $u$ ] = weiß **then** DFS-Visit( $u$ )

DFS-Visit( $u$ )

1. color[ $u$ ]  $\leftarrow$  grau
2. time  $\leftarrow$  time + 1;  $d[u] \leftarrow$  time
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.   **if** color[ $v$ ] = weiß **then**  $p[v] \leftarrow u$  ; DFS-Visit( $v$ )
5. color[ $u$ ]  $\leftarrow$  schwarz
6. time  $\leftarrow$  time + 1;  $f[u] \leftarrow$  time

time = 11



## Graphalgorithmen

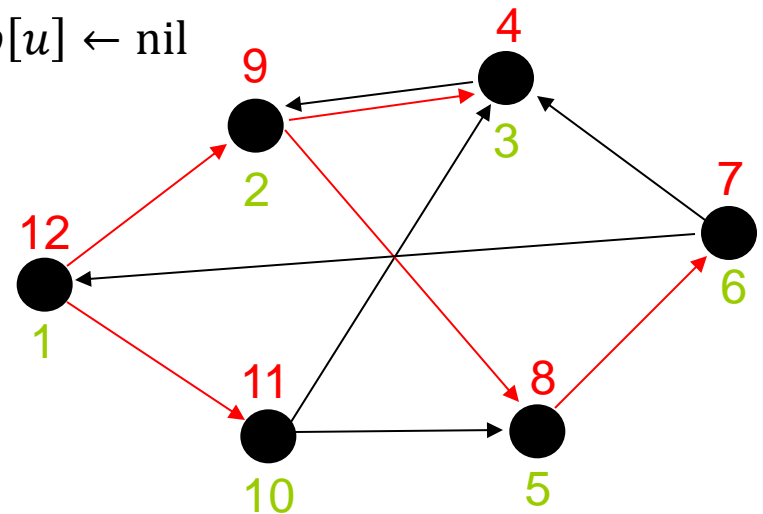
DFS( $G$ )

1. **for each** vertex  $u \in V$  **do** color[ $u$ ]  $\leftarrow$  weiß;  $p[u] \leftarrow$  nil
2. time  $\leftarrow$  0
3. **for each** vertex  $u \in V$  **do**
4.   **if** color[ $u$ ] = weiß **then** DFS-Visit( $u$ )

DFS-Visit( $u$ )

1. color[ $u$ ]  $\leftarrow$  grau
2. time  $\leftarrow$  time + 1;  $d[u] \leftarrow$  time
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.   **if** color[ $v$ ] = weiß **then**  $p[v] \leftarrow u$  ; DFS-Visit( $v$ )
5. color[ $u$ ]  $\leftarrow$  schwarz
6. time  $\leftarrow$  time + 1;  $f[u] \leftarrow$  time

time = 12



## Graphalgorithmen

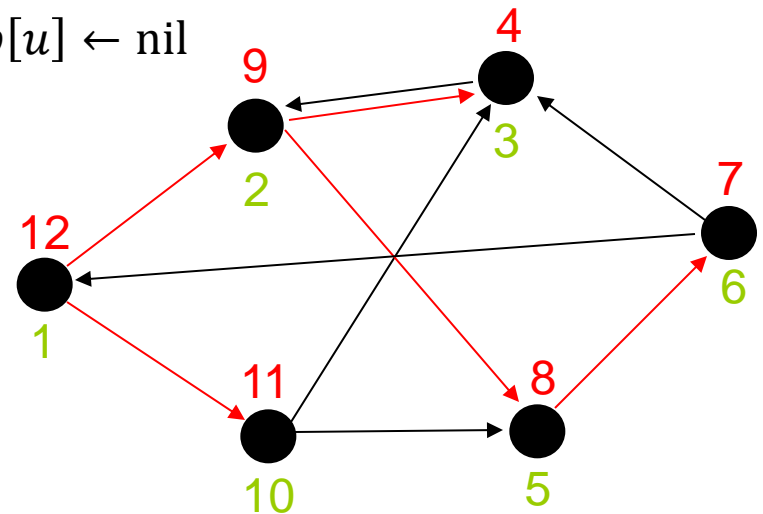
DFS( $G$ )

1. **for each** vertex  $u \in V$  **do** color[ $u$ ]  $\leftarrow$  weiß;  $p[u] \leftarrow$  nil
2. time  $\leftarrow$  0
3. **for each** vertex  $u \in V$  **do**
4.   **if** color[ $u$ ] = weiß **then** DFS-Visit( $u$ )

DFS-Visit( $u$ )

1. color[ $u$ ]  $\leftarrow$  grau
2. time  $\leftarrow$  time + 1;  $d[u] \leftarrow$  time
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.   **if** color[ $v$ ] = weiß **then**  $p[v] \leftarrow u$  ; DFS-Visit( $v$ )
5. color[ $u$ ]  $\leftarrow$  schwarz
6. time  $\leftarrow$  time + 1;  $f[u] \leftarrow$  time

time = 12



Laufzeit:  
 $O(|V| + |E|)$

## Graphalgorithmen

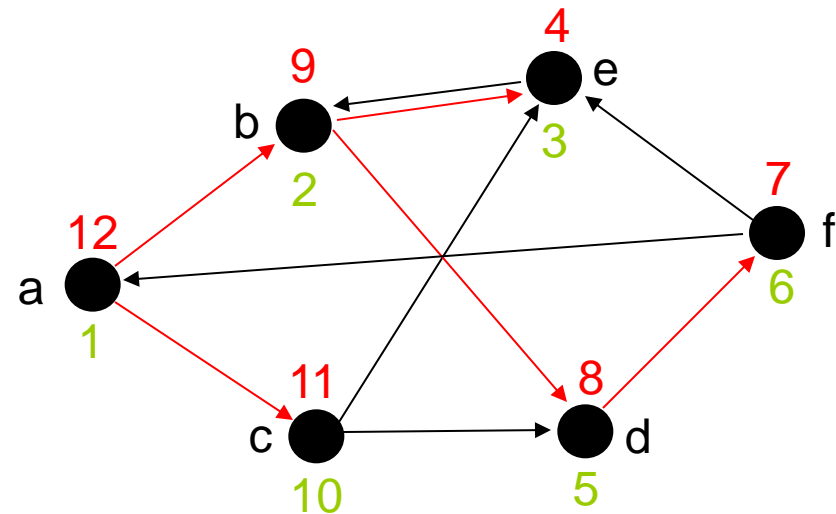
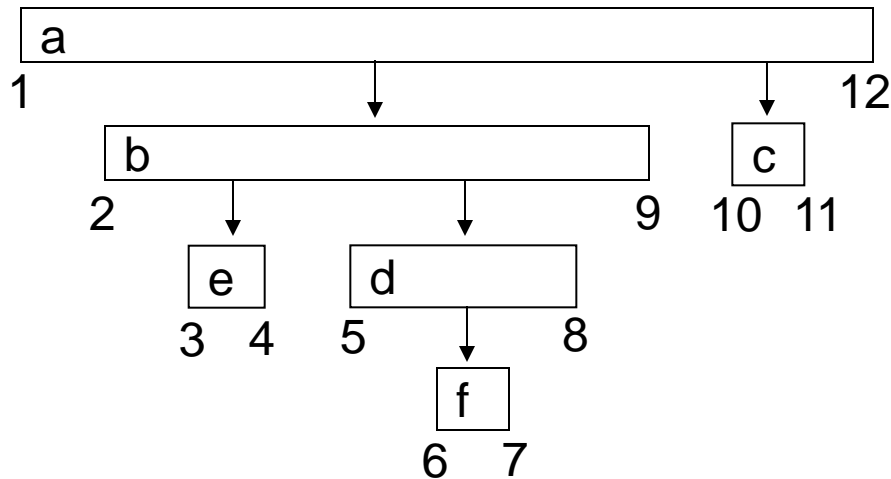
### *Satz 64 (Klammersatz zur Tiefensuche):*

In jeder Tiefensuche eines gerichteten oder ungerichteten Graphen gilt für jeden Knoten  $u$  und  $v$  genau eine der folgenden drei Bedingungen:

- Die Intervalle  $[d[u], f[u]]$  und  $[d[v], f[v]]$  sind vollständig disjunkt
- Intervall  $[d[u], f[u]]$  ist vollständig im Intervall  $[d[v], f[v]]$  enthalten und  $u$  ist Nachfolger von  $v$  im DFS-Wald
- Intervall  $[d[v], f[v]]$  ist vollständig im Intervall  $[d[u], f[u]]$  enthalten und  $v$  ist Nachfolger von  $u$  im DFS-Wald

# Graphalgorithmen

## Beispiel



## Graphalgorithmen

### *Beweis*

- Fall 1:  $d[u] < d[v]$ .

## Graphalgorithmen

### *Beweis*

- Fall 1:  $d[u] < d[v]$ .
- (a)  $d[v] < f[u]$ :
- $v$  wurde entdeckt als  $u$  noch grau war.



## Graphalgorithmen

### *Beweis*

- Fall 1:  $d[u] < d[v]$ .
- (a)  $d[v] < f[u]$ :
- $v$  wurde entdeckt als  $u$  noch grau war.
- $\Rightarrow v$  **Nachfolger von  $u$**

## Graphalgorithmen

### *Beweis*

- Fall 1:  $d[u] < d[v]$ .
- (a)  $d[v] < f[u]$ :
  - $v$  wurde entdeckt als  $u$  noch grau war.
  - $\Rightarrow v$  Nachfolger von  $u$
- Da  $v$  nach  $u$  entdeckt wurde, werden alle seine ausgehenden Kanten entdeckt und wird  $v$  abgearbeitet bevor die Suche zu  $u$  zurückkehrt und  $u$  abarbeitet

## Graphalgorithmen

### Beweis

- Fall 1:  $d[u] < d[v]$ .
- (a)  $d[v] < f[u]$ :
  - $v$  wurde entdeckt als  $u$  noch grau war.
  - $\Rightarrow v$  Nachfolger von  $u$
  - Da  $v$  nach  $u$  entdeckt wurde, werden alle seine ausgehenden Kanten entdeckt und wird  $v$  abgearbeitet bevor die Suche zu  $u$  zurückkehrt und  $u$  abarbeitet
- Daher ist  $[d[v], f[v]]$  in  $[d[u], f[u]]$  enthalten

## Graphalgorithmen

### Beweis

- Fall 1:  $d[u] < d[v]$ .
- (a)  $d[v] < f[u]$ :
  - $v$  wurde entdeckt als  $u$  noch grau war.
  - $\Rightarrow v$  Nachfolger von  $u$
  - Da  $v$  nach  $u$  entdeckt wurde, werden alle seine ausgehenden Kanten entdeckt und wird  $v$  abgearbeitet bevor die Suche zu  $u$  zurückkehrt und  $u$  abarbeitet
  - Daher ist  $[d[v], f[v]]$  in  $[d[u], f[u]]$  enthalten
- (b)  $f[u] < d[v]$ : Dann sind die Intervalle disjunkt

## Graphalgorithmen

### Beweis

- Fall 1:  $d[u] < d[v]$ .
- (a)  $d[v] < f[u]$ :
  - $v$  wurde entdeckt als  $u$  noch grau war.
  - $\Rightarrow v$  Nachfolger von  $u$
  - Da  $v$  nach  $u$  entdeckt wurde, werden alle seine ausgehenden Kanten entdeckt und wird  $v$  abgearbeitet bevor die Suche zu  $u$  zurückkehrt und  $u$  abarbeitet
  - Daher ist  $[d[v], f[v]]$  in  $[d[u], f[u]]$  enthalten
- (b)  $f[u] < d[v]$ : Dann sind die Intervalle disjunkt
- Fall 2: analog

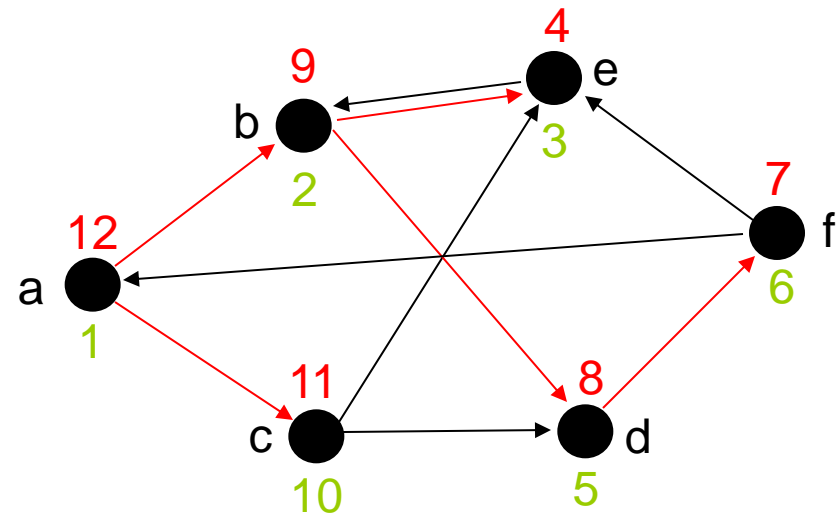
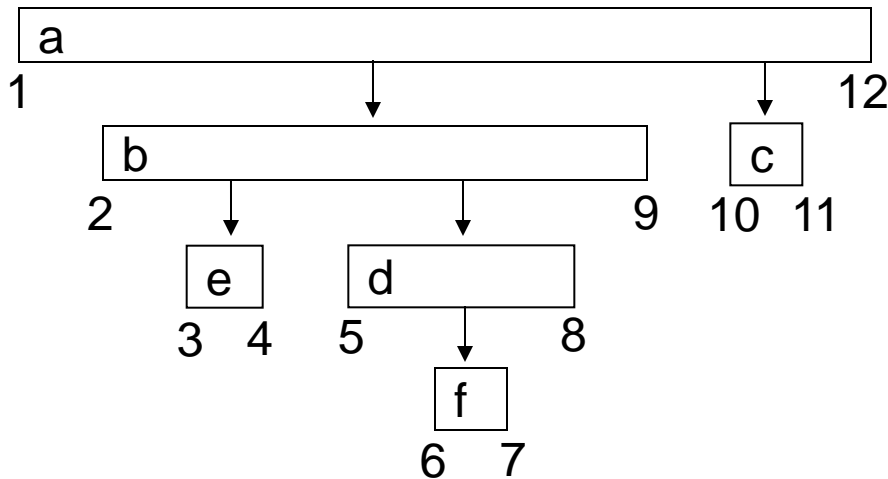
## Graphalgorithmen

### Beweis

- Fall 1:  $d[u] < d[v]$ .
- (a)  $d[v] < f[u]$ :
  - $v$  wurde entdeckt als  $u$  noch grau war.
  - $\Rightarrow v$  Nachfolger von  $u$
  - Da  $v$  nach  $u$  entdeckt wurde, werden alle seine ausgehenden Kanten entdeckt und wird  $v$  abgearbeitet bevor die Suche zu  $u$  zurückkehrt und  $u$  abarbeitet
  - Daher ist  $[d[v], f[v]]$  in  $[d[u], f[u]]$  enthalten
- (b)  $f[u] < d[v]$ : Dann sind die Intervalle disjunkt
- Fall 2: analog

## Graphalgorithmen

### Beispiel



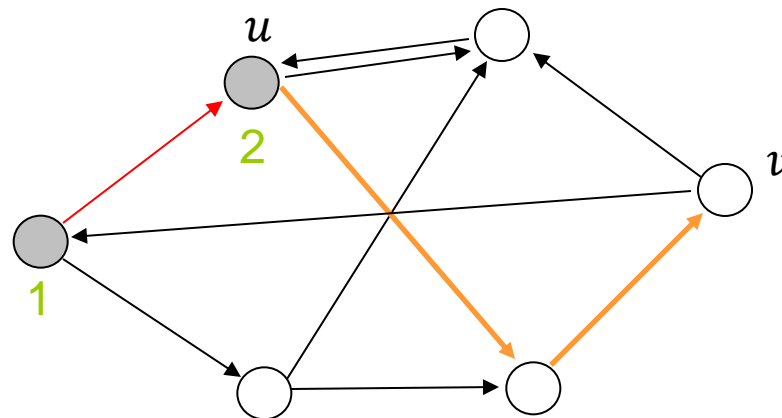
### Korollar 65

Knoten  $v$  ist echter ( $u \neq v$ ) Nachfolger von Knoten  $u$  im DFS-Wald von  $G$ ,  
gdw.  $d[u] < d[v] < f[v] < f[u]$ .

## Graphalgorithmen

### Satz 66 (Satz vom weißen Weg)

In einem DFS-Wald eines gerichteten oder ungerichteten Graph  $G$  ist Knoten  $v$  ein Nachfolger von Knoten  $u$ , gdw. zum Zeitpunkt  $d[u]$   $v$  über einen Pfad weißer Knoten erreicht werden kann.





## Graphalgorithmen

### *Satz 66 (Satz vom weißen Weg)*

In einem DFS-Wald eines gerichteten oder ungerichteten Graph  $G$  ist Knoten  $v$  ein Nachfolger von Knoten  $u$ , gdw. zum Zeitpunkt  $d[u]$   $v$  über einen Pfad weißer Knoten erreicht werden kann.

### *Beweis*

„ $\Rightarrow$ “ Annahme:  $v$  Nachfolger von  $u$  im DFS Wald

- Sei  $w$  beliebiger Knoten auf Pfad im DFS Wald von  $u$  nach  $v$
- Damit ist  $w$  Nachfolger von  $u$
- Nach Korollar 65:  $d[u] < d[w]$ . Somit ist  $w$  weiß zum Zeitpunkt  $d[u]$

## Graphalgorithmen

### *Satz 66 (Satz vom weißen Weg)*

In einem DFS-Wald eines gerichteten oder ungerichteten Graph  $G$  ist Knoten  $v$  ein Nachfolger von Knoten  $u$ , gdw. zum Zeitpunkt  $d[u]$   $v$  über einen Pfad weißer Knoten erreicht werden kann.

### *Beweis*

„ $\Leftarrow$ “ Annahme:  $v$  ist erreichbar von  $u$  über Pfad aus weißen Knoten zum Zeitpunkt  $d[u]$ , aber  $v$  wird nicht Nachfolger von  $u$  im DFS Wald

## Graphalgorithmen

### *Satz 66 (Satz vom weißen Weg)*

In einem DFS-Wald eines gerichteten oder ungerichteten Graph  $G$  ist Knoten  $v$  ein Nachfolger von Knoten  $u$ , gdw. zum Zeitpunkt  $d[u]$   $v$  über einen Pfad weißer Knoten erreicht werden kann.

### *Beweis*

„ $\Leftarrow$ “ Annahme:  $v$  ist erreichbar von  $u$  über Pfad aus weißen Knoten zum Zeitpunkt  $d[u]$ , aber  $v$  wird nicht Nachfolger von  $u$  im DFS Wald

- **ObdA.** sei  $v$  der einzige Knoten auf Pfad, der nicht Nachfolger wird

## Graphalgorithmen

### *Satz 66 (Satz vom weißen Weg)*

In einem DFS-Wald eines gerichteten oder ungerichteten Graph  $G$  ist Knoten  $v$  ein Nachfolger von Knoten  $u$ , gdw. zum Zeitpunkt  $d[u]$   $v$  über einen Pfad weißer Knoten erreicht werden kann.

### *Beweis*

„ $\Leftarrow$ “ Annahme:  $v$  ist erreichbar von  $u$  über Pfad aus weißen Knoten zum Zeitpunkt  $d[u]$ , aber  $v$  wird nicht Nachfolger von  $u$  im DFS Wald

- ObdA. sei  $v$  der einzige Knoten auf Pfad, der nicht Nachfolger wird
- Sei  $w$  der direkte Vorgänger von  $v$  auf dem Pfad und sei  $w$  Nachfolger von  $u$

## Graphalgorithmen

### *Satz 66 (Satz vom weißen Weg)*

In einem DFS-Wald eines gerichteten oder ungerichteten Graph  $G$  ist Knoten  $v$  ein Nachfolger von Knoten  $u$ , gdw. zum Zeitpunkt  $d[u]$   $v$  über einen Pfad weißer Knoten erreicht werden kann.

### *Beweis*

„ $\Leftarrow$ “ Annahme:  $v$  ist erreichbar von  $u$  über Pfad aus weißen Knoten zum Zeitpunkt  $d[u]$ , aber  $v$  wird nicht Nachfolger von  $u$  im DFS Wald

- ObdA. sei  $v$  der einzige Knoten auf Pfad, der nicht Nachfolger wird
- Sei  $w$  der direkte Vorgänger von  $v$  auf dem Pfad und sei  $w$  Nachfolger von  $u$
- **Korollar 65:  $f[w] \leq f[u]$**

## Graphalgorithmen

### *Satz 66 (Satz vom weißen Weg)*

In einem DFS-Wald eines gerichteten oder ungerichteten Graph  $G$  ist Knoten  $v$  ein Nachfolger von Knoten  $u$ , gdw. zum Zeitpunkt  $d[u]$   $v$  über einen Pfad weißer Knoten erreicht werden kann.

### *Beweis*

„ $\Leftarrow$ “ Annahme:  $v$  ist erreichbar von  $u$  über Pfad aus weißen Knoten zum Zeitpunkt  $d[u]$ , aber  $v$  wird nicht Nachfolger von  $u$  im DFS Wald

- ObdA. sei  $v$  der einzige Knoten auf Pfad, der nicht Nachfolger wird
- Sei  $w$  der direkte Vorgänger von  $v$  auf dem Pfad und sei  $w$  Nachfolger von  $u$
- Korollar 65:  $f[w] \leq f[u]$
- $v$  muss entdeckt werden, nachdem  $u$  entdeckt wurde und bevor  $w$  abgearbeitet ist, d.h.  $d[u] < d[v] < f[w] \leq f[u]$

## Graphalgorithmen

### Satz 66 (Satz vom weißen Weg)

In einem DFS-Wald eines gerichteten oder ungerichteten Graph  $G$  ist Knoten  $v$  ein Nachfolger von Knoten  $u$ , gdw. zum Zeitpunkt  $d[u]$   $v$  über einen Pfad weißer Knoten erreicht werden kann.

### Beweis

„ $\Leftarrow$ “ Annahme:  $v$  ist erreichbar von  $u$  über Pfad aus weißen Knoten zum Zeitpunkt  $d[u]$ , aber  $v$  wird nicht Nachfolger von  $u$  im DFS Wald

- ObdA. sei  $v$  der einzige Knoten auf Pfad, der nicht Nachfolger wird
- Sei  $w$  der direkte Vorgänger von  $v$  auf dem Pfad und sei  $w$  Nachfolger von  $u$
- Korollar 65:  $f[w] \leq f[u]$
- $v$  muss entdeckt werden, nachdem  $u$  entdeckt wurde und bevor  $w$  abgearbeitet ist, d.h.  $d[u] < d[v] < f[w] \leq f[u]$
- Somit ist  $[d[v], f[v]]$  in  $[d[u], f[u]]$  enthalten (Satz 64) und nach Korollar 65 ist  $v$  Nachfolger von  $u$

## Graphalgorithmen

### *Satz 66 (Satz vom weißen Weg)*

In einem DFS-Wald eines gerichteten oder ungerichteten Graph  $G$  ist Knoten  $v$  ein Nachfolger von Knoten  $u$ , gdw. zum Zeitpunkt  $d[u]$   $v$  über einen Pfad weißer Knoten erreicht werden kann.

### *Beweis*

„ $\Leftarrow$ “ Annahme:  $v$  ist erreichbar von  $u$  über Pfad aus weißen Knoten zum Zeitpunkt  $d[u]$ , aber  $v$  wird nicht Nachfolger von  $u$  im DFS Wald

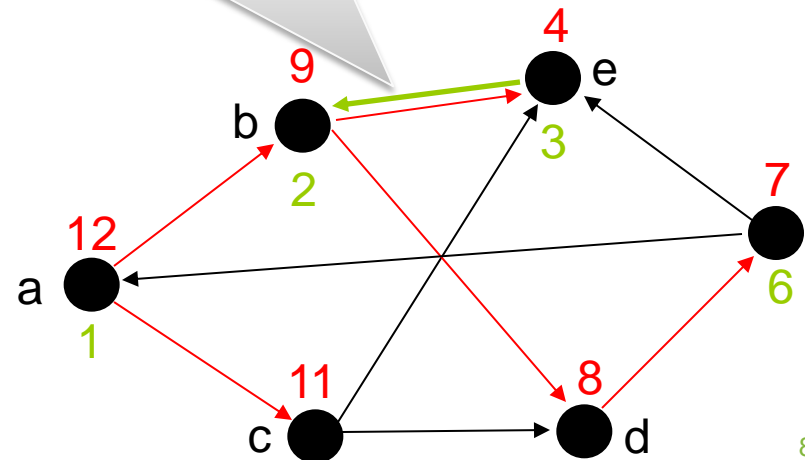
- ObdA. sei  $v$  der einzige Knoten auf Pfad, der nicht Nachfolger wird
- Sei  $w$  der direkte Vorgänger von  $v$  auf dem Pfad und sei  $w$  Nachfolger von  $u$
- Korollar 65:  $f[w] \leq f[u]$
- $v$  muss entdeckt werden, nachdem  $u$  entdeckt wurde und bevor  $w$  abgearbeitet ist, d.h.  $d[u] < d[v] < f[w] \leq f[u]$
- Somit ist  $[d[v], f[v]]$  in  $[d[u], f[u]]$  enthalten (Satz 64) und nach Korollar 65 ist  $v$  Nachfolger von  $u$



## Graphalgorithmen

### Klassifikation von Kanten

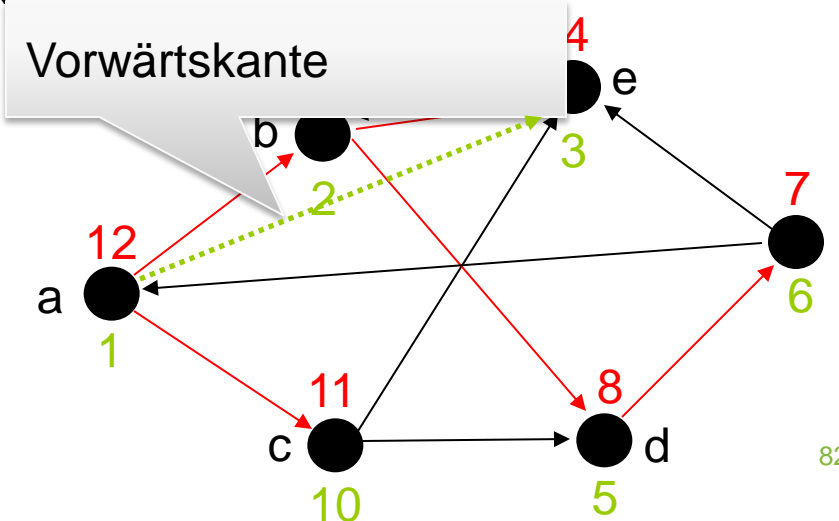
- **Baumkanten** sind Kanten des DFS-Walds  $G$
- **Rückwärtskanten** sind Kanten  $(u, v)$ , die Knoten  $u$  mit Vorgängern von  $u$  im DFS-Baum verbinden
- **Vorwärtskanten** sind die nicht-Baum Kanten  $(u, v)$ , die  $u$  mit einem Nachfolger  $v$  in einem DFS-Baum verbinden
- **Kreuzungskanten** sind alle übrigen Kanten



## Graphalgorithmen

### Klassifikation von Kanten

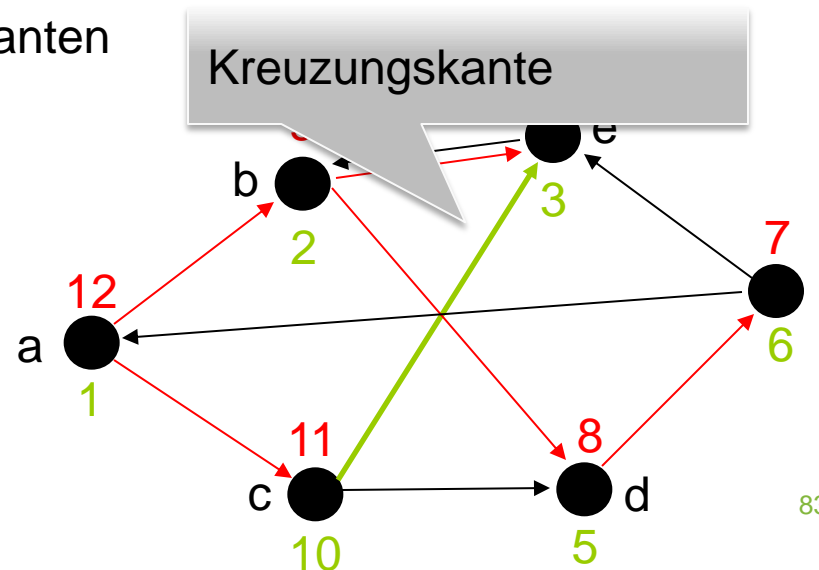
- **Baumkanten** sind Kanten des DFS-Walds  $G$
- **Rückwärtskanten** sind Kanten  $(u, v)$ , die Knoten  $u$  mit Vorgängern von  $u$  im DFS-Baum verbinden
- **Vorwärtskanten** sind die nicht-Baum Kanten  $(u, v)$ , die  $u$  mit einem Nachfolger  $v$  in einem DFS-Baum verbinden
- **Kreuzungskanten** sind alle übrigen Kanten



## Graphalgorithmen

### Klassifikation von Kanten

- **Baumkanten** sind Kanten des DFS-Walds  $G$
- **Rückwärtskanten** sind Kanten  $(u, v)$ , die Knoten  $u$  mit Vorgängern von  $u$  im DFS-Baum verbinden
- **Vorwärtskanten** sind die nicht-Baum Kanten  $(u, v)$ , die  $u$  mit einem Nachfolger  $v$  in einem DFS-Baum verbinden
- **Kreuzungskanten** sind alle übrigen Kanten



## Graphalgorithmen

### *Problem*

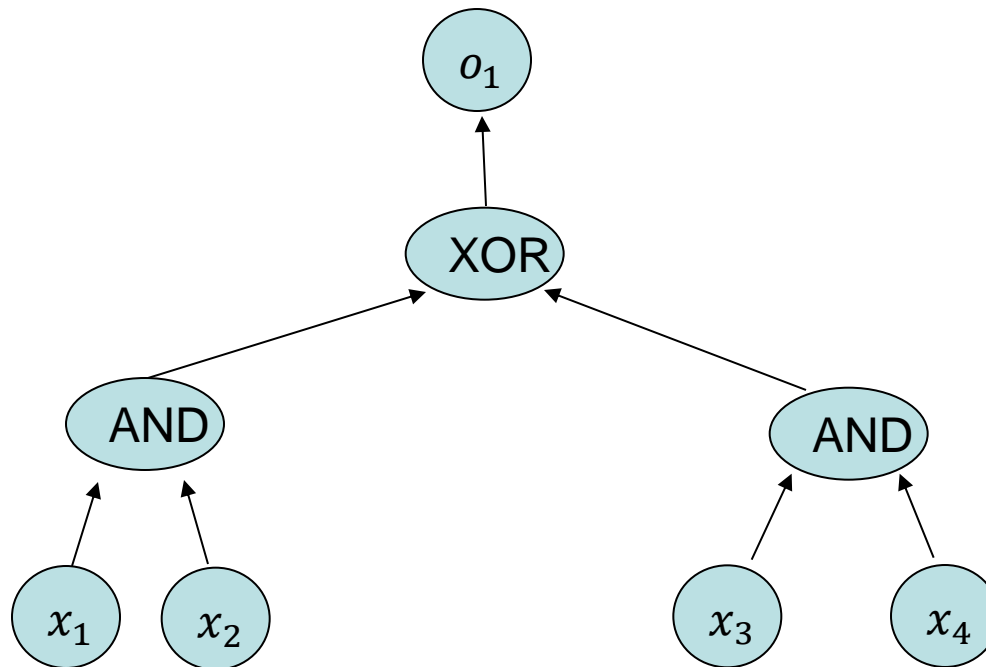
- Eingabe: Schaltkreis (ohne Zyklen) mit NOT, AND, OR, XOR Gattern,  $n$  Eingänge und  $m$  Ausgänge sowie eine Belegung der Eingänge durch boolesche Werte
- Ausgabe: Die Ausgabewerte des Schaltkreises bei dieser Belegung

### *Bemerkung*

Wir nehmen an, dass der Schaltkreis als gerichteter Graph gegeben ist. Jeder Knoten entspricht einem Gatter.

## Graphalgorithmen

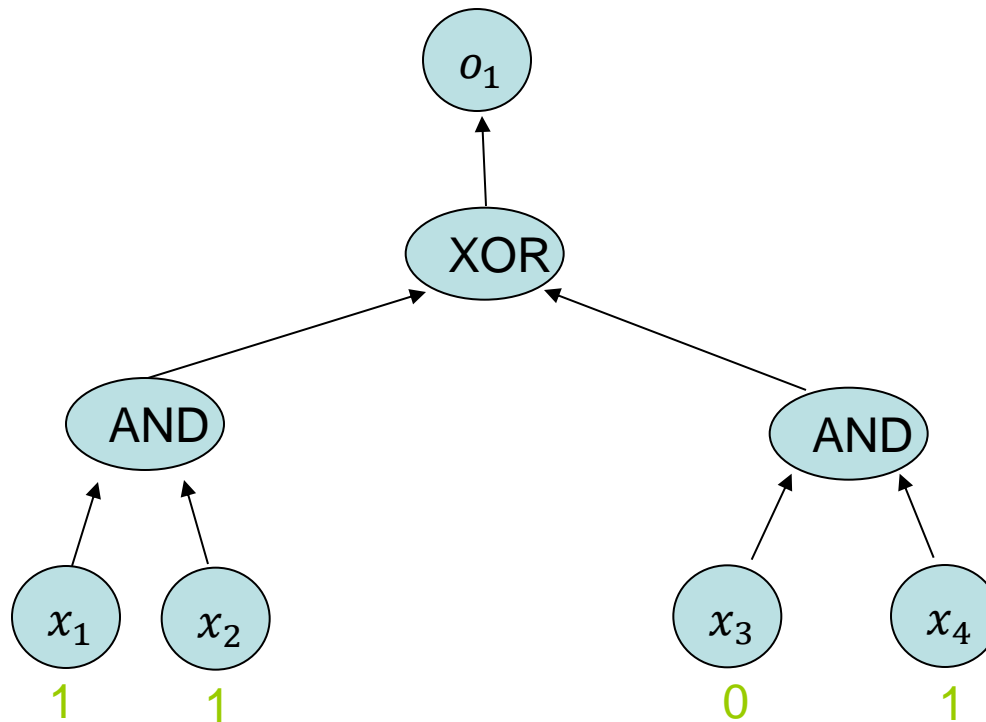
*Beispiel:*



## Graphalgorithmen

### Beispiel:

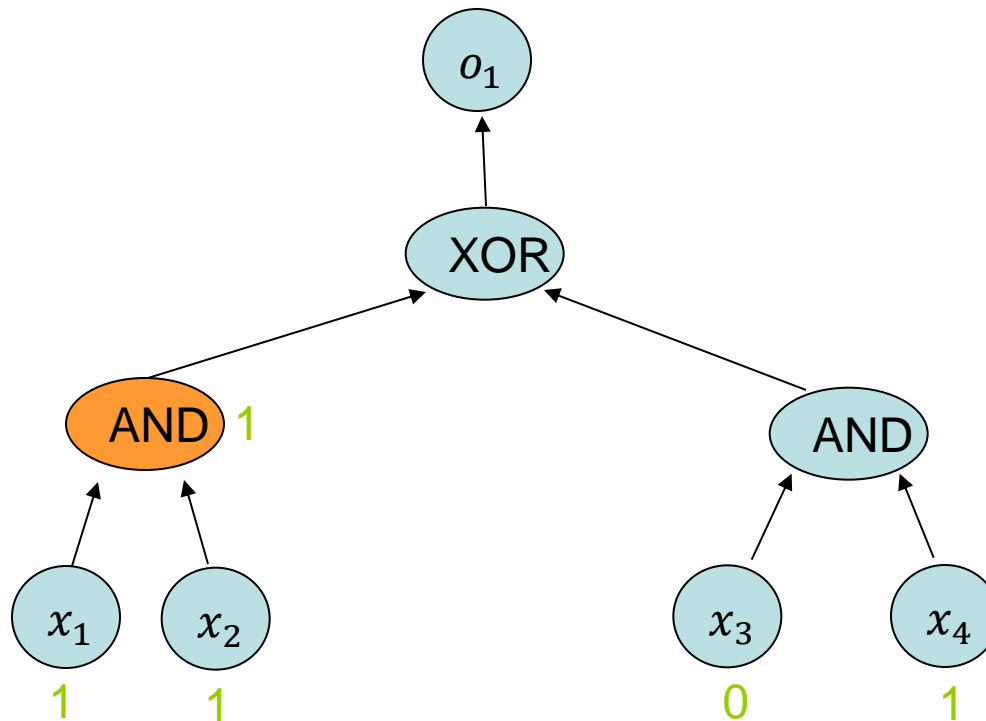
- Eingabe: 1,1,0,1



## Graphalgorithmen

### Beispiel:

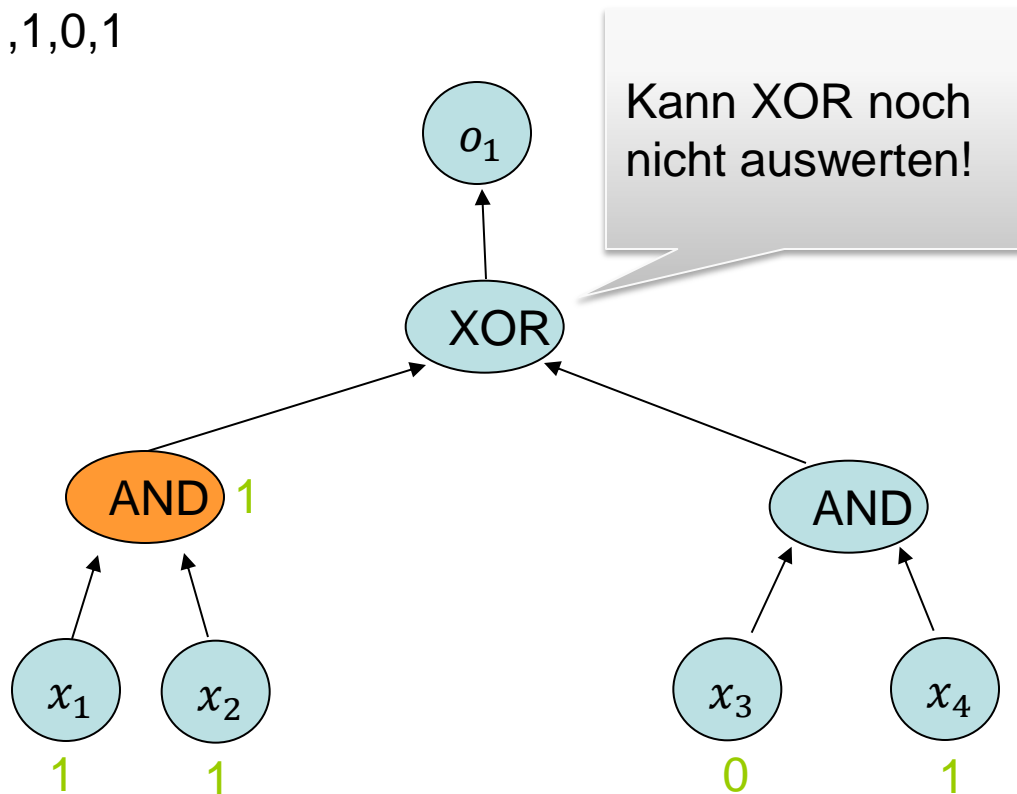
- Eingabe: 1,1,0,1



## Graphalgorithmen

### Beispiel:

- Eingabe: 1,1,0,1

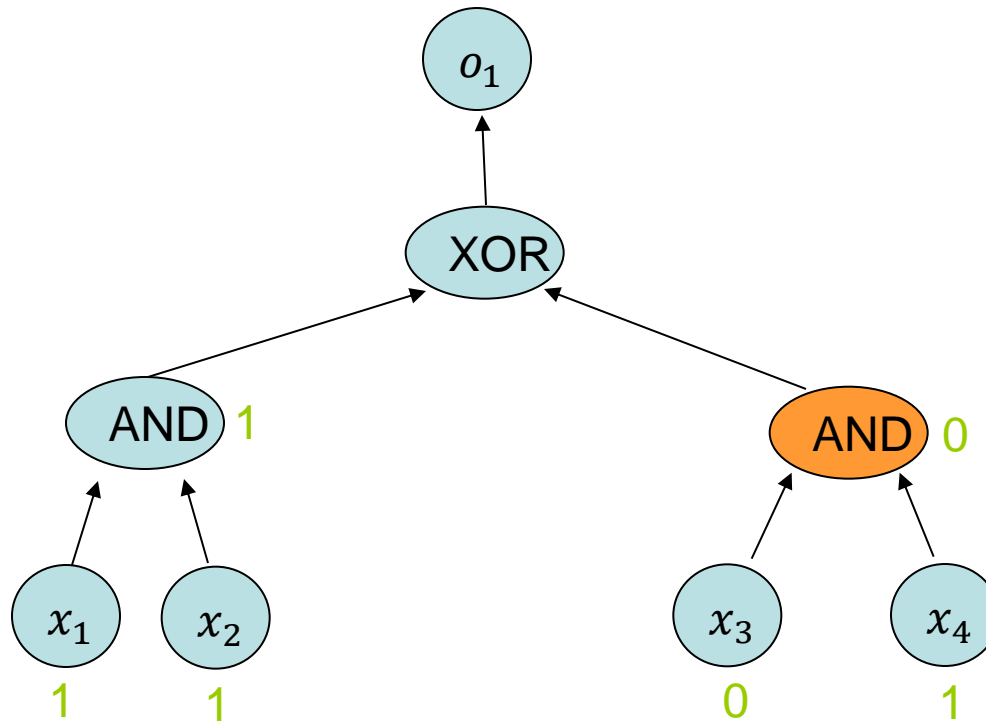




## Graphalgorithmen

### Beispiel:

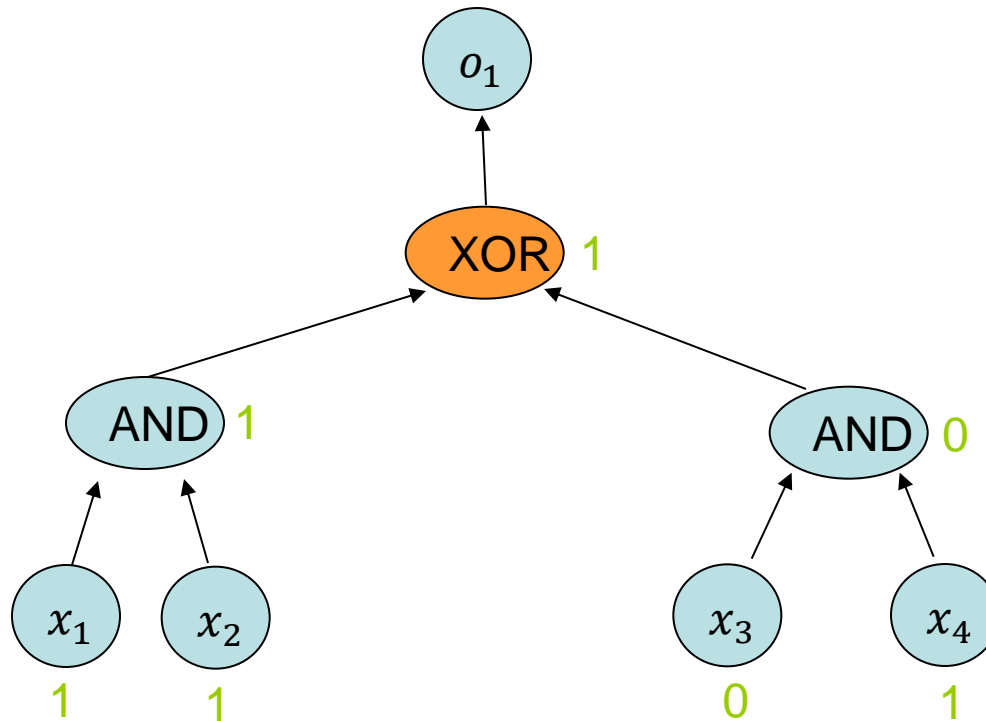
- Eingabe: 1,1,0,1



## Graphalgorithmen

### Beispiel:

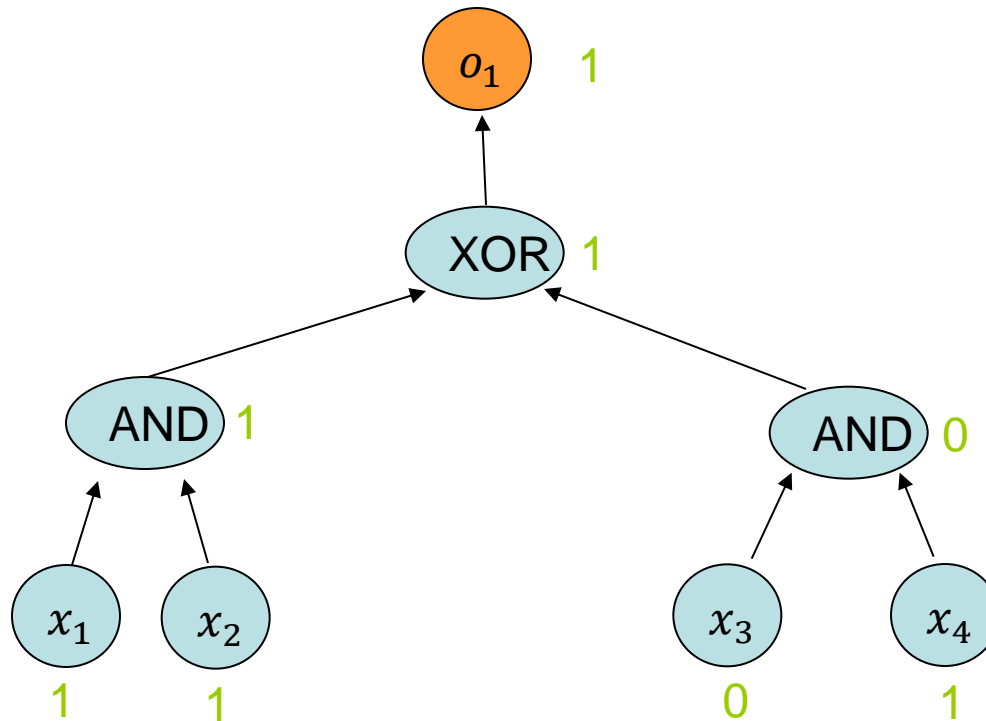
- Eingabe: 1,1,0,1



## Graphalgorithmen

### Beispiel:

- Eingabe: 1,1,0,1



## Graphalgorithmen

### *Frage*

Wie kann man eine Reihenfolge der Knoten berechnen, so dass man an jedem Knoten sofort den Wert des Gatters ausrechnen kann?

### *Eine Möglichkeit*

- Topologisches Sortieren
- Sortierung eines gerichteten, azyklischen Graphen, so dass für jede Kante  $(u, v)$   $u$  in der Sortierung vor  $v$  steht („ $u$  wird vor  $v$  ausgewertet“)

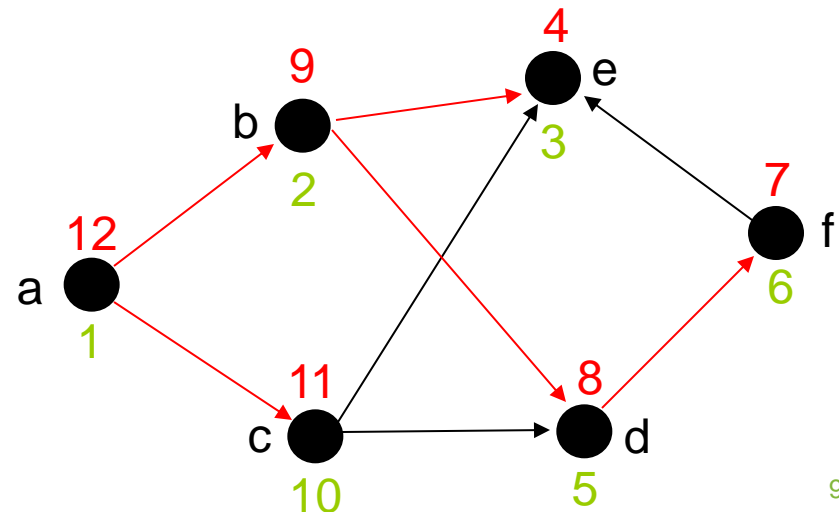
## Graphalgorithmen

### Topsort( $G$ )

1. Rufe DFS( $G$ ) auf, um Zeitstempel  $f[v]$  für jeden Knoten  $v$  zu berechnen
2. Sobald ein Knoten abgearbeitet ist, füge ihn zu Beginn einer Liste  $L$  ein (absteigende Sortierung nach  $f[v]$ -Werten)
3. **return**  $L$

### Sortierung im Beispiel

- a, c, b, d, f, e



## Graphalgorithmen

### Topsort( $G$ )

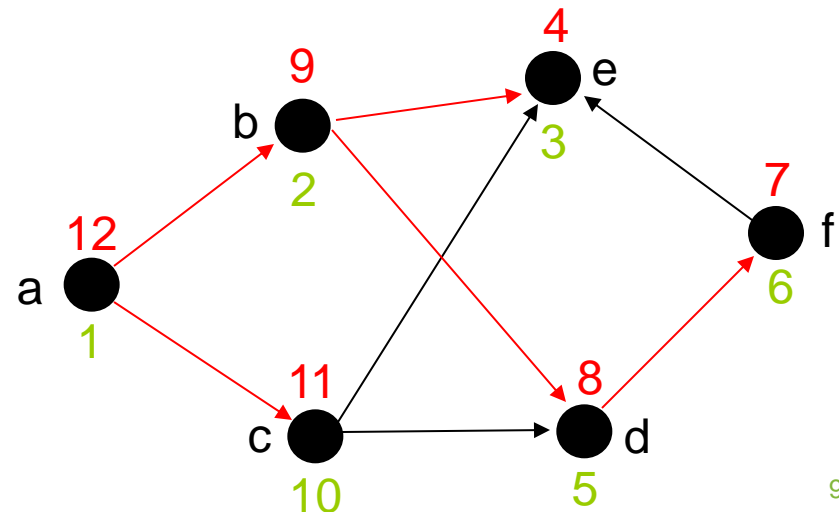
1. Rufe  $\text{DFS}(G)$  auf, um Zeitstempel  $f[v]$  für jeden Knoten  $v$  zu berechnen
2. Sobald ein Knoten abgearbeitet ist, füge ihn zu Beginn einer Liste  $L$  ein (absteigende Sortierung nach  $f[v]$ -Werten)
3. **return**  $L$

### Sortierung im Beispiel

- a, c, b, d, f, e

### Laufzeit

- $\mathcal{O}(|V| + |E|)$



## Graphalgorithmen

### *Lemma 67*

Ein gerichteter Graph  $G$  ist azyklisch, gdw. eine Tiefensuche in  $G$  keine Rückwärtskanten hat.

### *Beweis*

„ $\Rightarrow$ “: Annahme:  $G$  azyklisch und es gibt Rückwärtskante  $(u, v)$ . Dann ist  $v$  direkter Vorgänger von  $u$  im DFS Wald. Es gibt also einen Weg von  $v$  nach  $u$  und die Rückwärtskante schließt den Zyklus. (Widerspruch zur Azyklizität)

## Graphalgorithmen

### Lemma 67

Ein gerichteter Graph  $G$  ist azyklisch, gdw. eine Tiefensuche in  $G$  keine Rückwärtskanten hat.

### Beweis

„ $\Rightarrow$ “: Annahme:  $G$  azyklisch und es gibt Rückwärtskante  $(u, v)$ . Dann ist  $v$  direkter Vorgänger von  $u$  im DFS Wald. Es gibt also einen Weg von  $v$  nach  $u$  und die Rückwärtskante schließt den Zyklus. (Widerspruch zur Azyklizität)

„ $\Leftarrow$ “: Annahme: Tiefensuche hat keine Rückwärtskanten und  $G$  enthält Zyklus  $C$ . Sei  $v$  der erste Knoten aus  $C$ , der entdeckt wurde und sei  $(u, v)$  die vorhergehende Kante im Zyklus  $C$ . Zum Zeitpunkt  $d[v]$  gibt es einen weißen Pfad von  $v$  nach  $u$ . Nach Satz 64 wird  $u$  Nachfolger von  $v$  und  $(u, v)$  somit Rückwärtskante. (Widerspruch!)



## Graphalgorithmen

### *Lemma 67*

Ein gerichteter Graph  $G$  ist azyklisch, gdw. eine Tiefensuche in  $G$  keine Rückwärtskanten hat.

### *Beweis*

„ $\Rightarrow$ “: Annahme:  $G$  azyklisch und es gibt Rückwärtskante  $(u, v)$ . Dann ist  $v$  direkter Vorgänger von  $u$  im DFS Wald. Es gibt also einen Weg von  $v$  nach  $u$  und die Rückwärtskante schließt den Zyklus. (Widerspruch zur Azyklizität)

„ $\Leftarrow$ “: Annahme: Tiefensuche hat keine Rückwärtskanten und  $G$  enthält Zyklus  $C$ . Sei  $v$  der erste Knoten aus  $C$ , der entdeckt wurde und sei  $(u, v)$  die vorhergehende Kante im Zyklus  $C$ . Zum Zeitpunkt  $d[v]$  gibt es einen weißen Pfad von  $v$  nach  $u$ . Nach Satz 64 wird  $u$  Nachfolger von  $v$  und  $(u, v)$  somit Rückwärtskante. (Widerspruch!)

## Graphalgorithmen

### Satz 68

$\text{TopSort}(G)$  berechnet eine topologische Sortierung eines gerichteten azyklischen Graphen.

## Graphalgorithmen

### Satz 68

$\text{TopSort}(G)$  berechnet eine topologische Sortierung eines gerichteten azyklischen Graphen.

### Beweis

- Annahme: Wir führen DFS auf gerichtetem Graph aus, um die Abarbeitungszeitpunkte zu bestimmen.

## Graphalgorithmen

### Satz 68

TopSort( $G$ ) berechnet eine topologische Sortierung eines gerichteten azyklischen Graphen.

### Beweis

- Annahme: Wir führen DFS auf gerichtetem Graph aus, um die Abarbeitungszeitpunkte zu bestimmen.
- Z.Z.: Für jedes Paar von Knoten  $u, v \in V$  mit  $u \neq v$  gilt: Gibt es Kante von  $u$  nach  $v$ , so ist  $f[v] < f[u]$ .

## Graphalgorithmen

### Satz 68

TopSort( $G$ ) berechnet eine topologische Sortierung eines gerichteten azyklischen Graphen.

### Beweis

- Annahme: Wir führen DFS auf gerichtetem Graph aus, um die Abarbeitungszeitpunkte zu bestimmen.
- Z.Z.: Für jedes Paar von Knoten  $u, v \in V$  mit  $u \neq v$  gilt: Gibt es Kante von  $u$  nach  $v$ , so ist  $f[v] < f[u]$ .
- Betrachte Zeitpunkt, wenn eine beliebige Kante  $(u, v)$  durch Tiefensuche entdeckt wird. Dann kann  $v$  nicht grau sein, denn dann wäre  $(u, v)$  Rückwärtskante und  $G$  nach Lemma 67 nicht azyklisch.

## Graphalgorithmen

### Satz 68

TopSort( $G$ ) berechnet eine topologische Sortierung eines gerichteten azyklischen Graphen.

### Beweis

- Annahme: Wir führen DFS auf gerichtetem Graph aus, um die Abarbeitungszeitpunkte zu bestimmen.
- Z.Z.: Für jedes Paar von Knoten  $u, v \in V$  mit  $u \neq v$  gilt: Gibt es Kante von  $u$  nach  $v$ , so ist  $f[v] < f[u]$ .
- Betrachte Zeitpunkt, wenn eine beliebige Kante  $(u, v)$  durch Tiefensuche entdeckt wird. Dann kann  $v$  nicht grau sein, denn dann wäre  $(u, v)$  Rückwärtskante und  $G$  nach Lemma 67 nicht azyklisch.
- **Daher ist  $v$  entweder weiß oder schwarz.**

## Graphalgorithmen

### Satz 68

TopSort( $G$ ) berechnet eine topologische Sortierung eines gerichteten azyklischen Graphen.

### Beweis

- Annahme: Wir führen DFS auf gerichtetem Graph aus, um die Abarbeitungszeitpunkte zu bestimmen.
- Z.Z.: Für jedes Paar von Knoten  $u, v \in V$  mit  $u \neq v$  gilt: Gibt es Kante von  $u$  nach  $v$ , so ist  $f[v] < f[u]$ .
- Betrachte Zeitpunkt, wenn eine beliebige Kante  $(u, v)$  durch Tiefensuche entdeckt wird. Dann kann  $v$  nicht grau sein, denn dann wäre  $(u, v)$  Rückwärtskante und  $G$  nach Lemma 67 nicht azyklisch.
- Daher ist  $v$  entweder weiß oder schwarz.
- Ist  $v$  weiß, so wird  $v$  Nachfolger von  $u$  und es gilt  $f[v] < f[u]$

## Graphalgorithmen

### Satz 68

TopSort( $G$ ) berechnet eine topologische Sortierung eines gerichteten azyklischen Graphen.

### Beweis

- Annahme: Wir führen DFS auf gerichtetem Graph aus, um die Abarbeitungszeitpunkte zu bestimmen.
- Z.Z.: Für jedes Paar von Knoten  $u, v \in V$  mit  $u \neq v$  gilt: Gibt es Kante von  $u$  nach  $v$ , so ist  $f[v] < f[u]$ .
- Betrachte Zeitpunkt, wenn eine beliebige Kante  $(u, v)$  durch Tiefensuche entdeckt wird. Dann kann  $v$  nicht grau sein, denn dann wäre  $(u, v)$  Rückwärtskante und  $G$  nach Lemma 67 nicht azyklisch.
- Daher ist  $v$  entweder weiß oder schwarz.
- Ist  $v$  weiß, so wird  $v$  Nachfolger von  $u$  und es gilt  $f[v] < f[u]$
- Ist  $v$  schwarz, dann gilt erst recht  $f[v] < f[u]$ . Damit folgt der Satz.



## Graphalgorithmen

### Satz 68

TopSort( $G$ ) berechnet eine topologische Sortierung eines gerichteten azyklischen Graphen.

### Beweis

- Annahme: Wir führen DFS auf gerichtetem Graph aus, um die Abarbeitungszeitpunkte zu bestimmen.
- Z.Z.: Für jedes Paar von Knoten  $u, v \in V$  mit  $u \neq v$  gilt: Gibt es Kante von  $u$  nach  $v$ , so ist  $f[v] < f[u]$ .
- Betrachte Zeitpunkt, wenn eine beliebige Kante  $(u, v)$  durch Tiefensuche entdeckt wird. Dann kann  $v$  nicht grau sein, denn dann wäre  $(u, v)$  Rückwärtskante und  $G$  nach Lemma 67 nicht azyklisch.
- Daher ist  $v$  entweder weiß oder schwarz.
- Ist  $v$  weiß, so wird  $v$  Nachfolger von  $u$  und es gilt  $f[v] < f[u]$
- Ist  $v$  schwarz, dann gilt erst recht  $f[v] < f[u]$ . Damit folgt der Satz.

# Graphalgorithmen

## *Zusammenfassung*

- Tiefensuche bietet andere Möglichkeit (neben Breitensuche) zur Graphtraversierung in Laufzeit  $\mathbf{O}(|V| + |E|)$
- Die Sortierung der Tiefensuche kann zum topologischen Sortieren benutzt werden