

# Algorithmen und Datenstrukturen

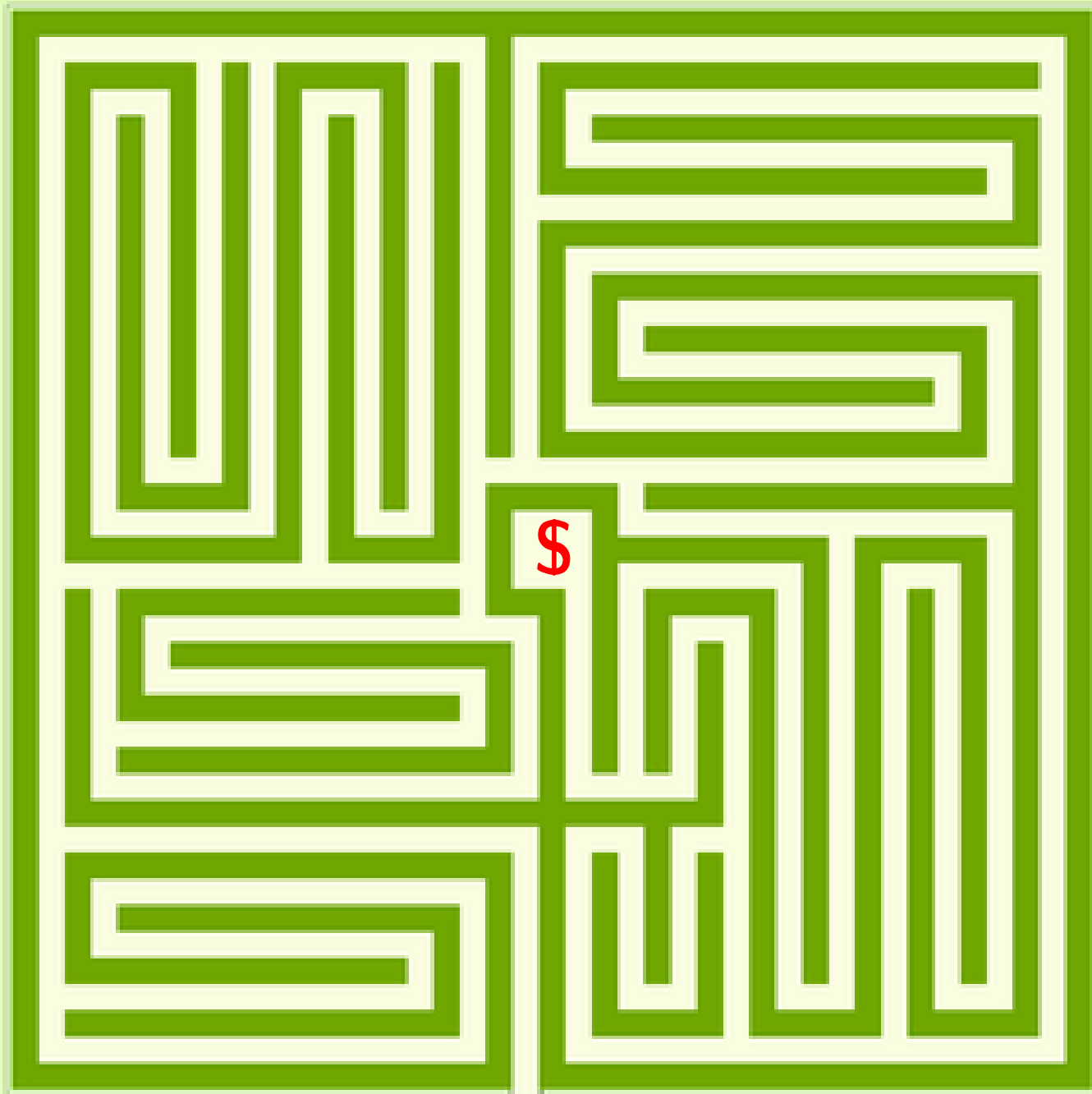
Wintersemester 2022/23

20. Vorlesung

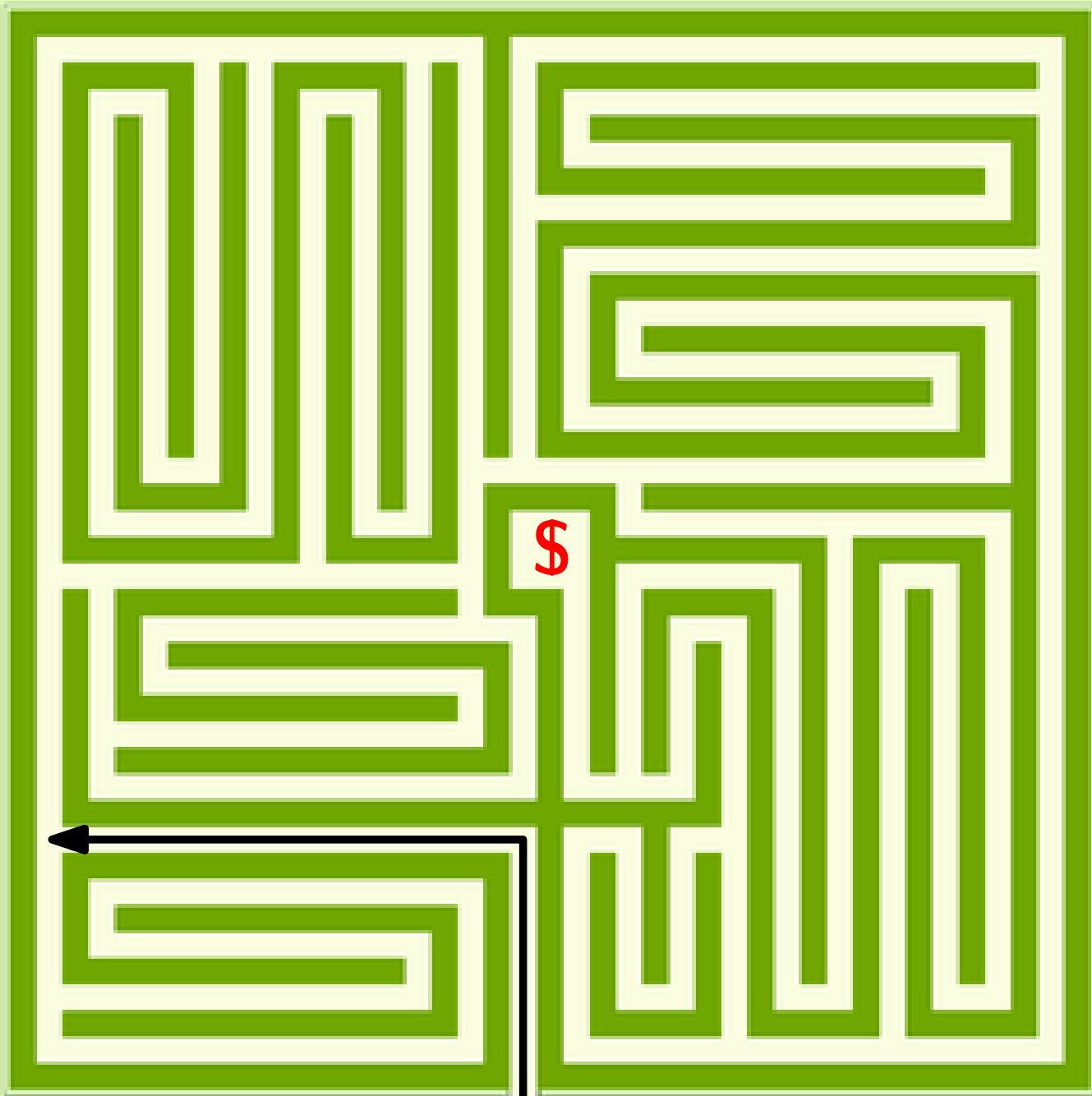
## Tiefensuche und topologische Sortierung

# ADS-Repetitorium

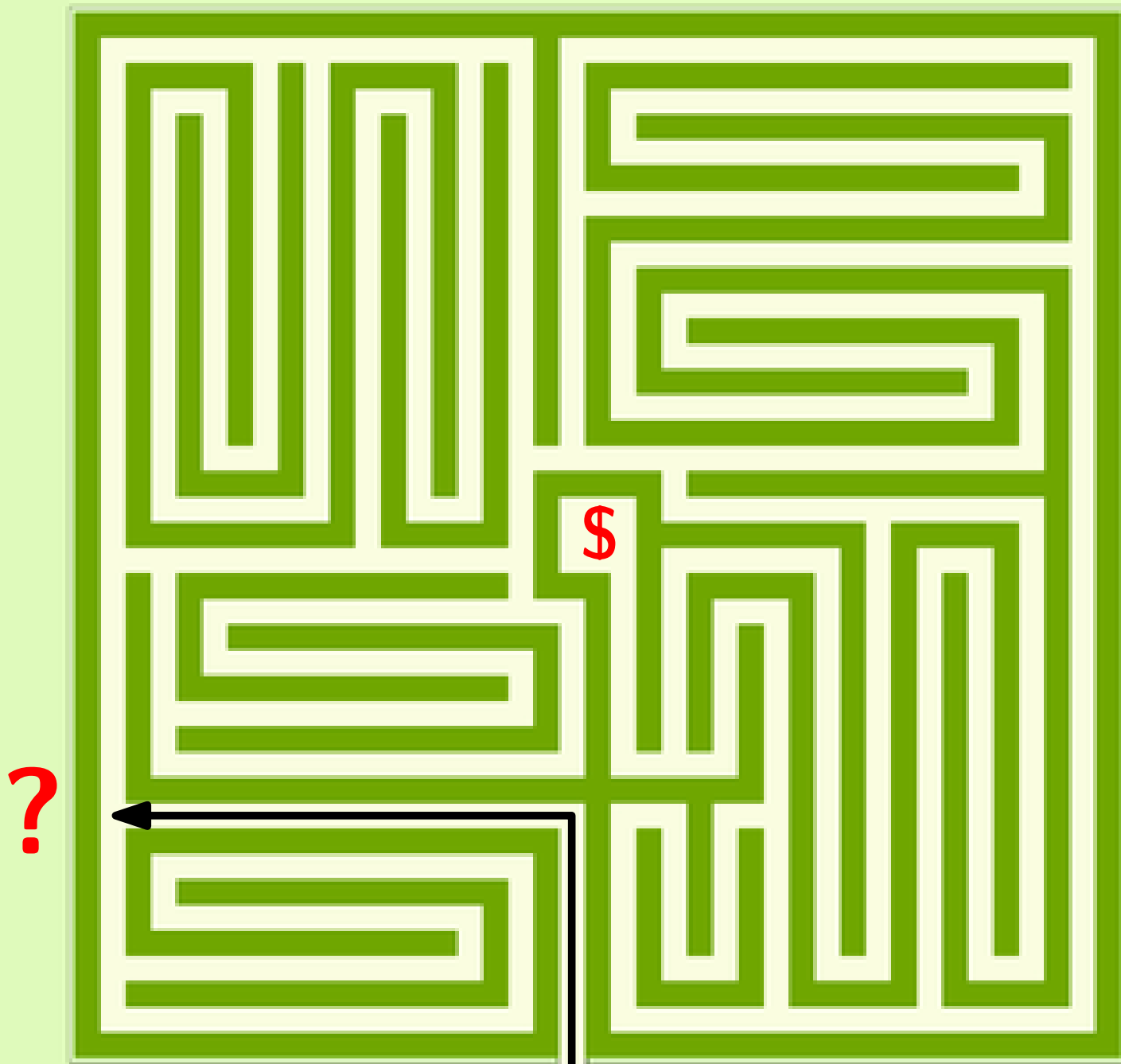
- Zur Vorbereitung auf die Nachklausur am 14.04.2023.
- Mo, 27.3., bis Fr, 31.03., jeweils 9–15 Uhr (Pause 12–13)
- Leitung: Felix Hauser (Ü6) & Samuel Wolf (Ü1)
- Kurze Zusammenfassung des Stoffs
- Durchrechnen von Übungsaufgaben und einer Probeklausur
- Der Notenbonus vom WS gilt auch bei der Nachklausur!



„Maze-01 Grüningen hedge maze 1576 (destroyed)“  
von RTH – Eigenes Werk. Lizenziert unter CC BY-SA 3.0 über Wikimedia Commons



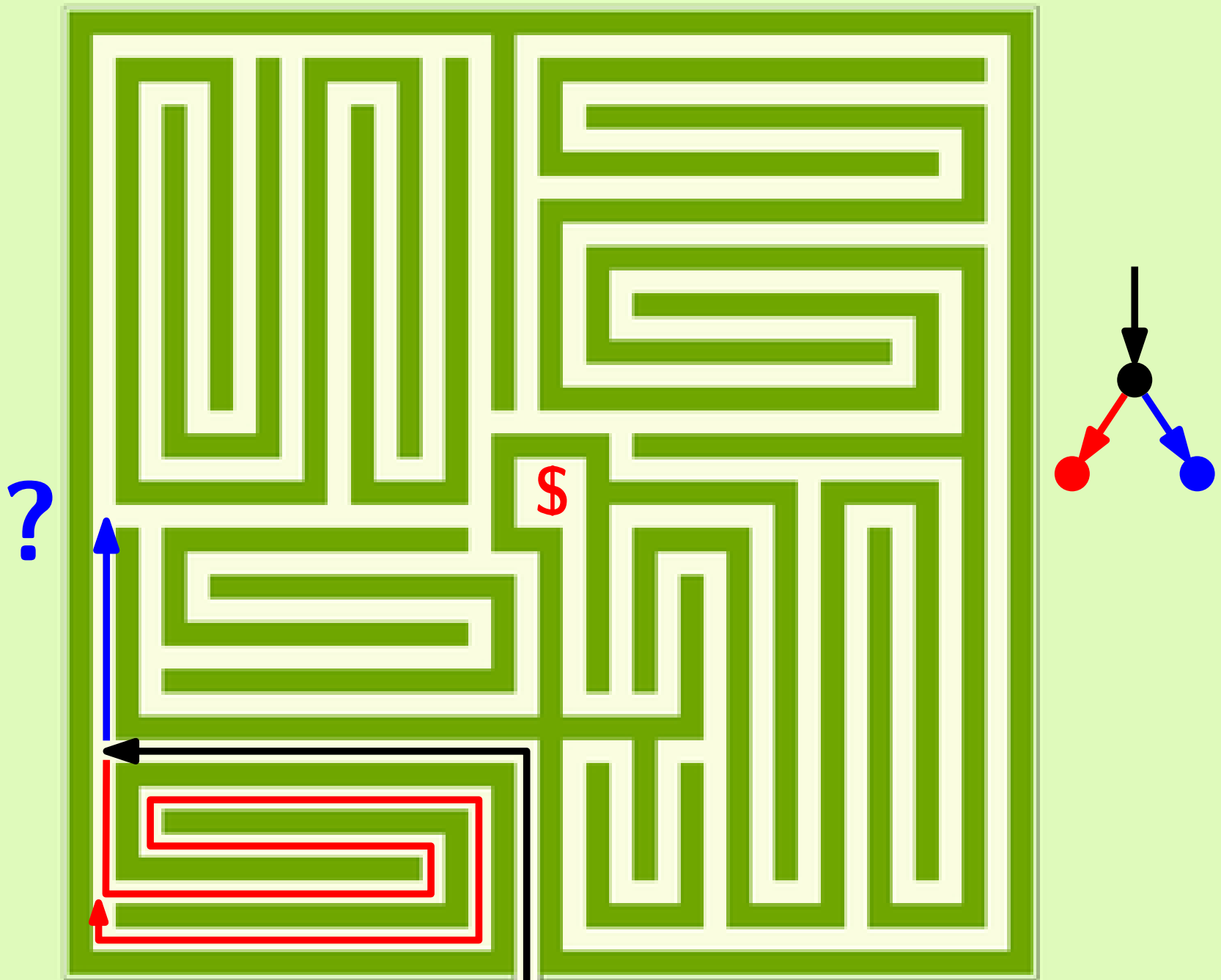
„Maze-01 Grüningen hedge maze 1576 (destroyed)“  
von RTH – Eigenes Werk. Lizenziert unter CC BY-SA 3.0 über Wikimedia Commons



„Maze-01 Grüningen hedge maze 1576 (destroyed)“  
von RTH – Eigenes Werk. Lizenziert unter CC BY-SA 3.0 über Wikimedia Commons



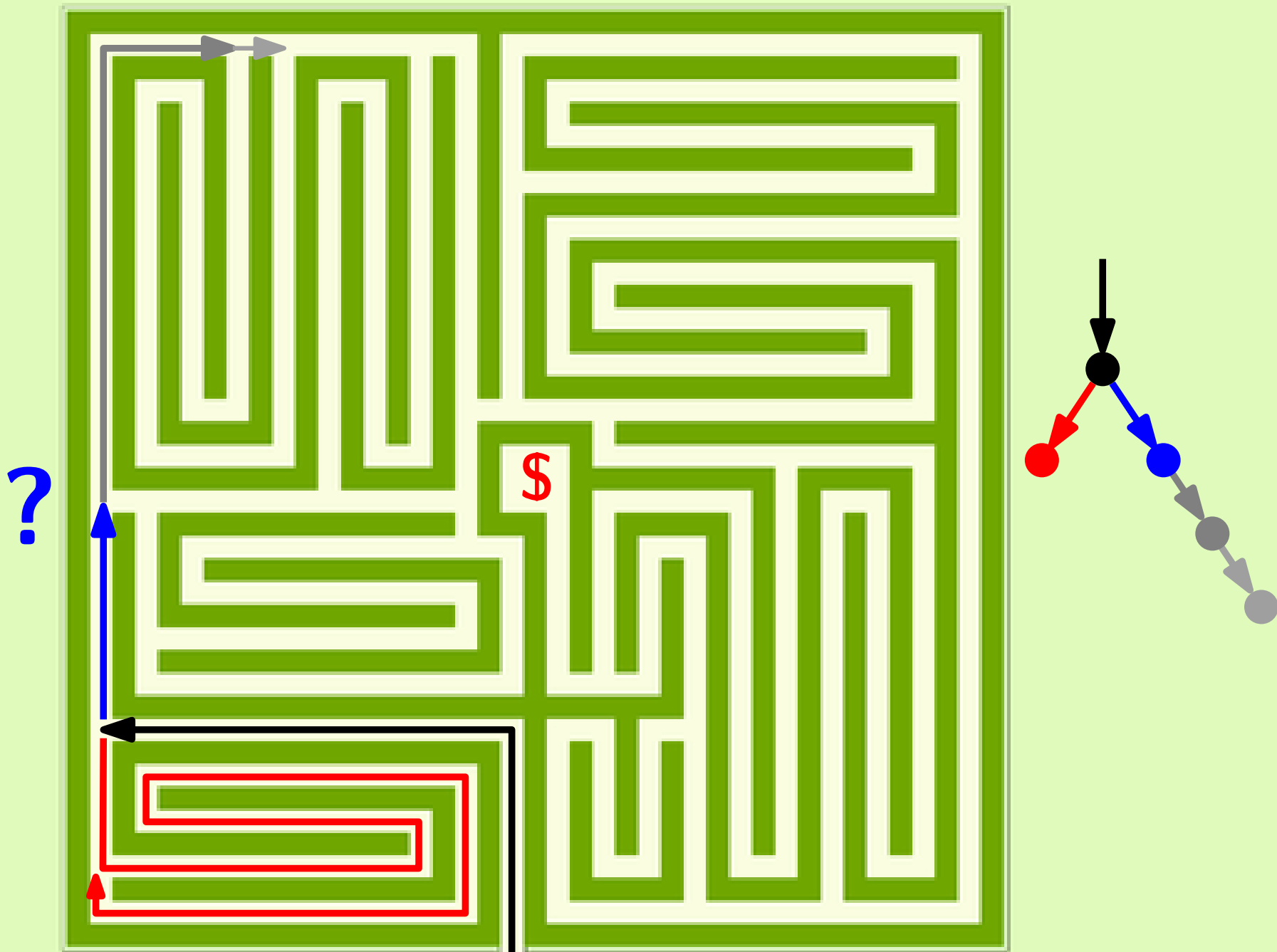




„Maze-01 Grüningen hedge maze 1576 (destroyed)“  
von RTH – Eigenes Werk. Lizenziert unter CC BY-SA 3.0 über Wikimedia Commons



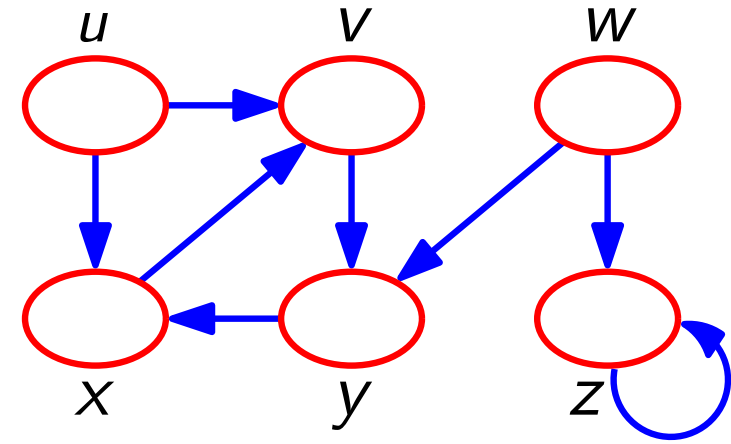




„Maze-01 Grüningen hedge maze 1576 (destroyed)“  
von RTH – Eigenes Werk. Lizenziert unter CC BY-SA 3.0 über Wikimedia Commons

# Tiefensuche

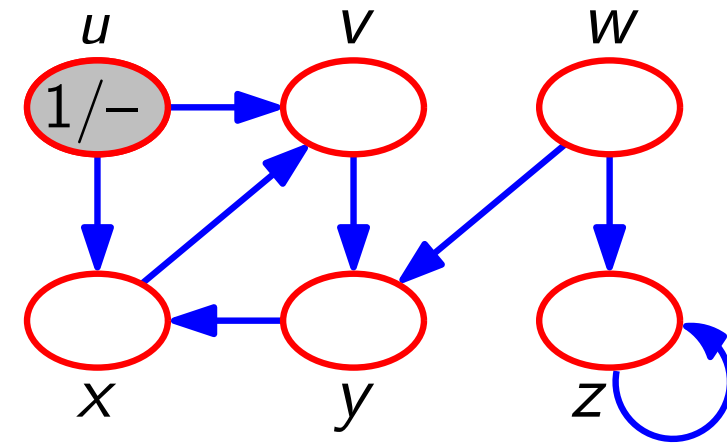
Eingabe: (un)gerichteter Graph  $G$



# Tiefensuche

Eingabe: (un)gerichteter Graph  $G$

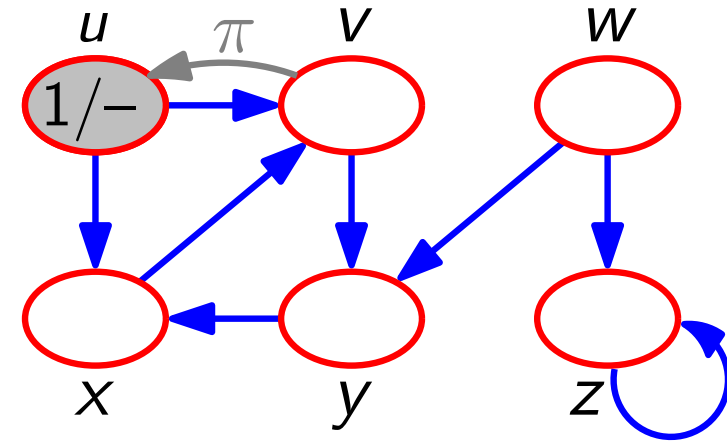
Ausgabe: – Besuchsintervalle ( $u.d/u.f$ )  
discovery time finish time



# Tiefensuche

Eingabe: (un)gerichteter Graph  $G$

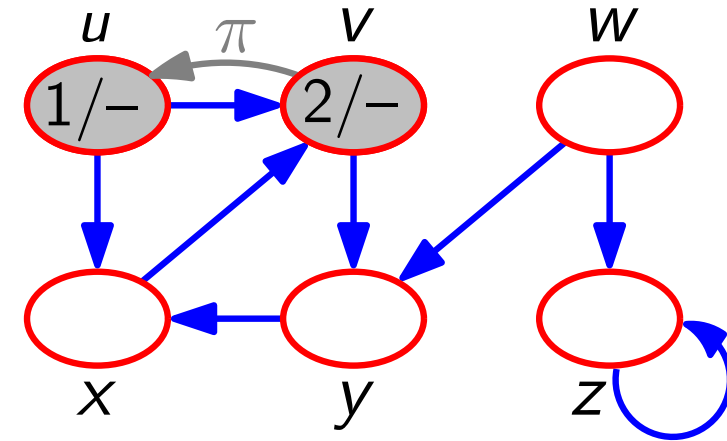
Ausgabe: – Besuchsintervalle ( $u.d / u.f$ )  
 – DFS-Wald ( $\leftarrow \pi$ )



# Tiefensuche

Eingabe: (un)gerichteter Graph  $G$

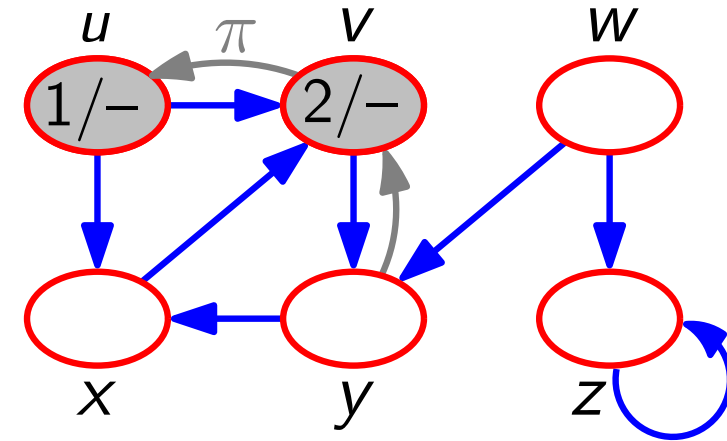
Ausgabe: – Besuchsintervalle ( $u.d/u.f$ )  
 – DFS-Wald ( $\leftarrow \pi$ )



# Tiefensuche

Eingabe: (un)gerichteter Graph  $G$

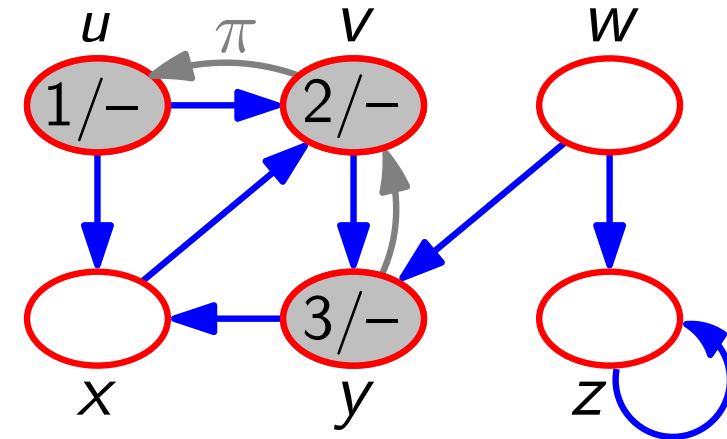
Ausgabe: – Besuchsintervalle  $(u.d / u.f)$   
 – DFS-Wald  $(\leftarrow \pi)$



# Tiefensuche

Eingabe: (un)gerichteter Graph  $G$

Ausgabe: – Besuchsintervalle  $(u.d / u.f)$   
 – DFS-Wald  $(\leftarrow \pi)$

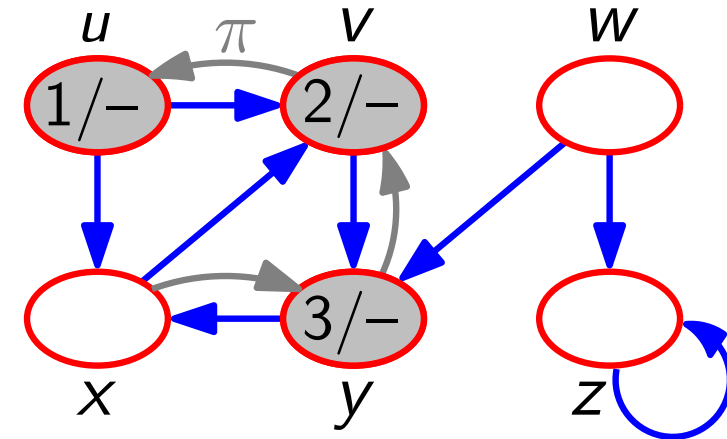




# Tiefensuche

Eingabe: (un)gerichteter Graph  $G$

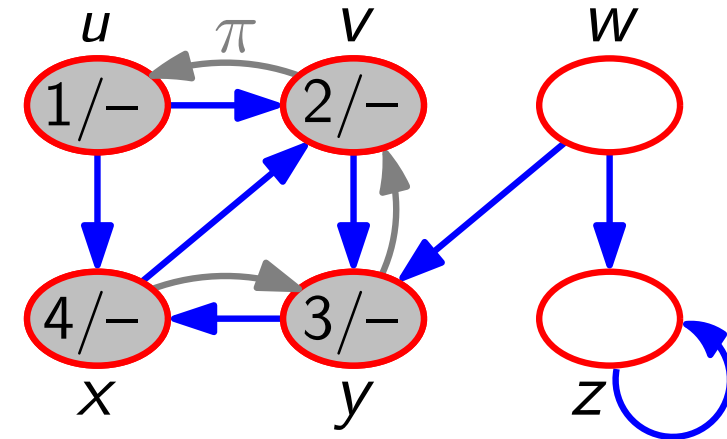
Ausgabe: – Besuchsintervalle  $(u.d / u.f)$   
 – DFS-Wald  $(\leftarrow \pi)$



# Tiefensuche

Eingabe: (un)gerichteter Graph  $G$

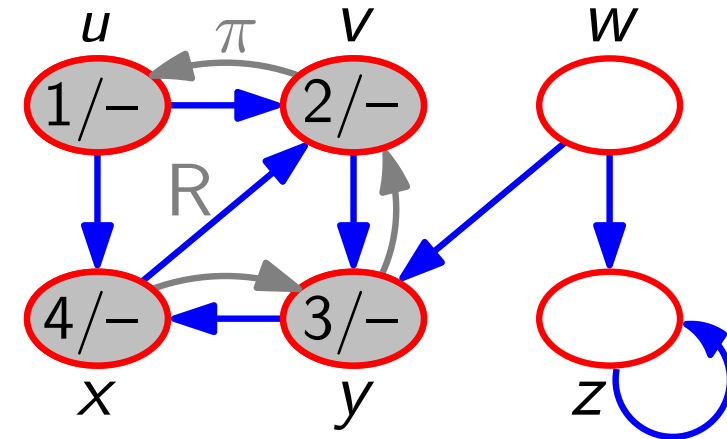
Ausgabe: – Besuchsintervalle ( $u.d/u.f$ )  
 – DFS-Wald ( $\leftarrow \pi$ )



# Tiefensuche

Eingabe: (un)gerichteter Graph  $G$

Ausgabe: – Besuchsintervalle ( $u.d/u.f$ )  
 – DFS-Wald ( $\leftarrow \pi$ )



# Tiefensuche

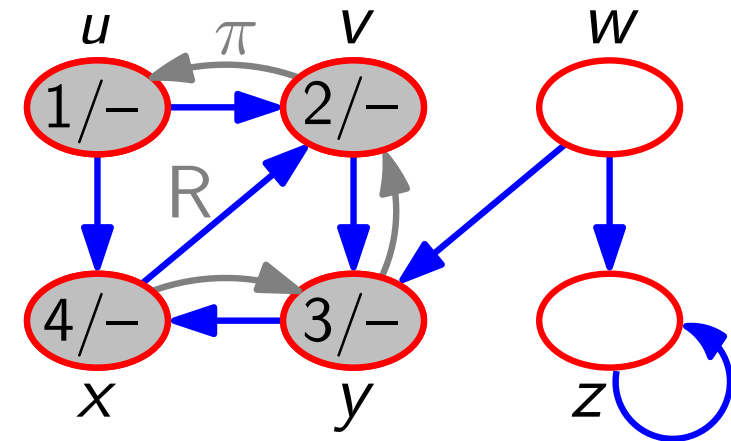
Eingabe: (un)gerichteter Graph  $G$

Ausgabe:

- Besuchsintervalle ( $u.d/u.f$ )  
discovery time finish time
- DFS-Wald ( $\leftarrow \pi$ )

– Klassifizierung der Graphkanten:

- Baumkanten (Kanten von  $G_\pi$ )
- Rückwärtskanten (R)



*Farbe Zielknoten:*

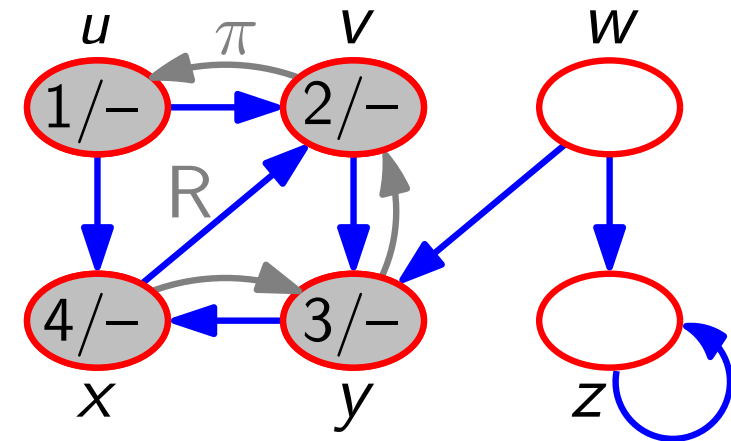
# Tiefensuche

Eingabe: (un)gerichteter Graph  $G$

Ausgabe: – Besuchsintervalle ( $u.d/u.f$ )  
discovery time finish time  
 – DFS-Wald ( $\leftarrow \pi$ )

– Klassifizierung der Graphkanten:

- Baumkanten (Kanten von  $G_\pi$ )
- Rückwärtskanten (R)



*Farbe Zielknoten:*

*weiss*

*grau*

# Tiefensuche

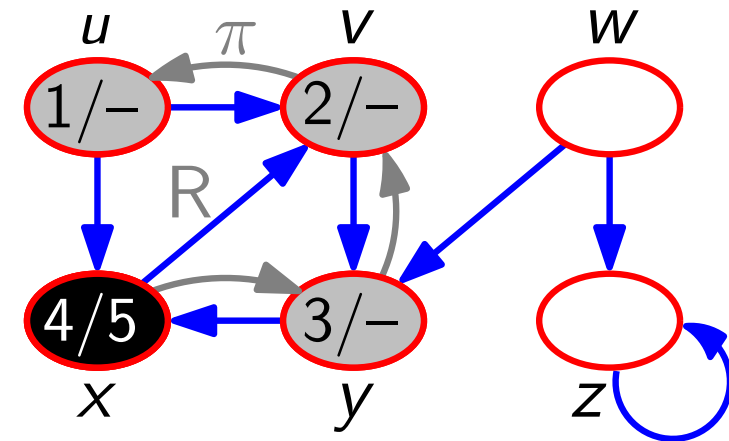
Eingabe: (un)gerichteter Graph  $G$

Ausgabe:

- Besuchsintervalle ( $u.d/u.f$ )  
discovery time finish time
- DFS-Wald ( $\leftarrow \pi$ )

– Klassifizierung der Graphkanten:

- Baumkanten (Kanten von  $G_\pi$ )
- Rückwärtskanten (R)



*Farbe Zielknoten:*

*weiss*

*grau*

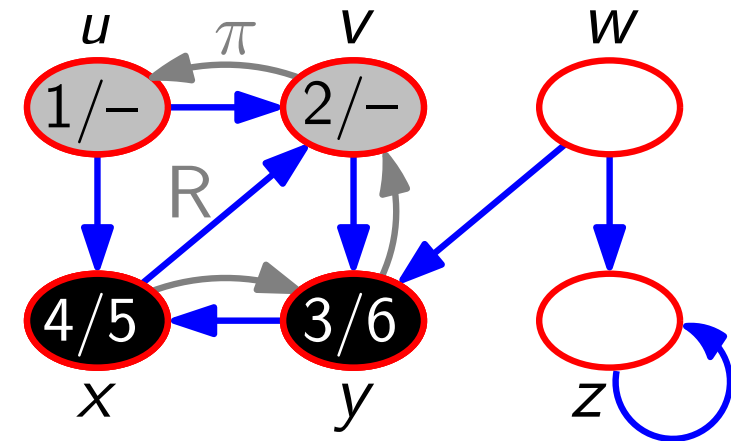
# Tiefensuche

Eingabe: (un)gerichteter Graph  $G$

Ausgabe: – Besuchsintervalle ( $u.d/u.f$ )  
 – DFS-Wald ( $\leftarrow \pi$ )

– Klassifizierung der Graphkanten:

- Baumkanten (Kanten von  $G_\pi$ )
- Rückwärtskanten (R)



Farbe Zielknoten:

weiss

grau

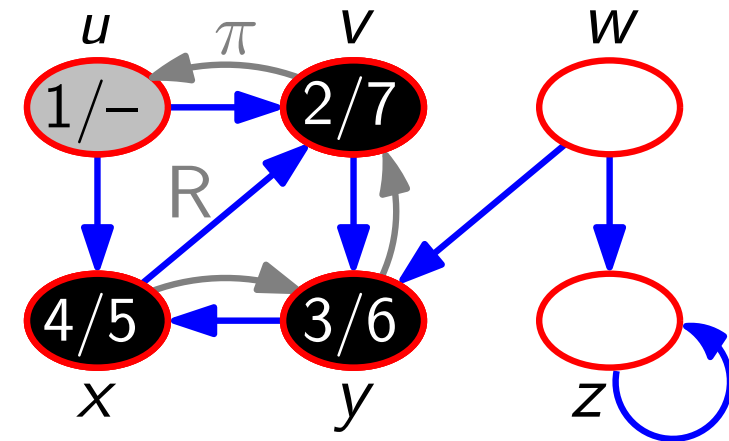
# Tiefensuche

Eingabe: (un)gerichteter Graph  $G$

Ausgabe: – Besuchsintervalle ( $u.d/u.f$ )  
 – DFS-Wald ( $\leftarrow \pi$ )

– Klassifizierung der Graphkanten:

- Baumkanten (Kanten von  $G_\pi$ )
- Rückwärtskanten (R)



*Farbe Zielknoten:*

*weiss*

*grau*



# Tiefensuche

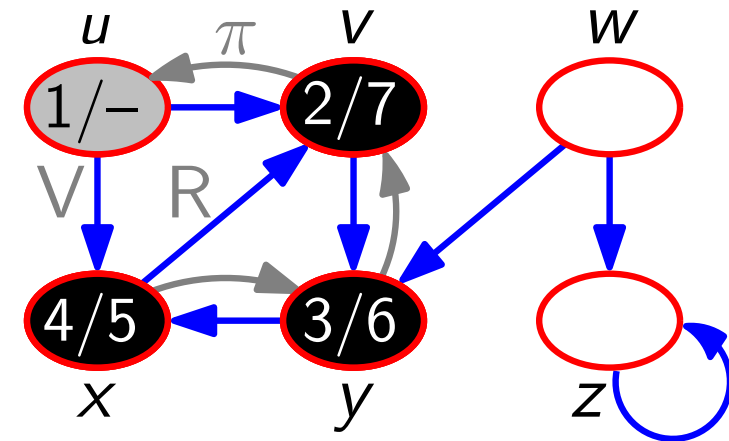
Eingabe: (un)gerichteter Graph  $G$

Ausgabe: – Besuchsintervalle ( $u.d/u.f$ )  
discovery time finish time

– DFS-Wald ( $\leftarrow \pi$ )

– Klassifizierung der Graphkanten:

- Baumkanten (Kanten von  $G_\pi$ )
- Rückwärtskanten (R)



*Farbe Zielknoten:*

*weiss*

*grau*

# Tiefensuche

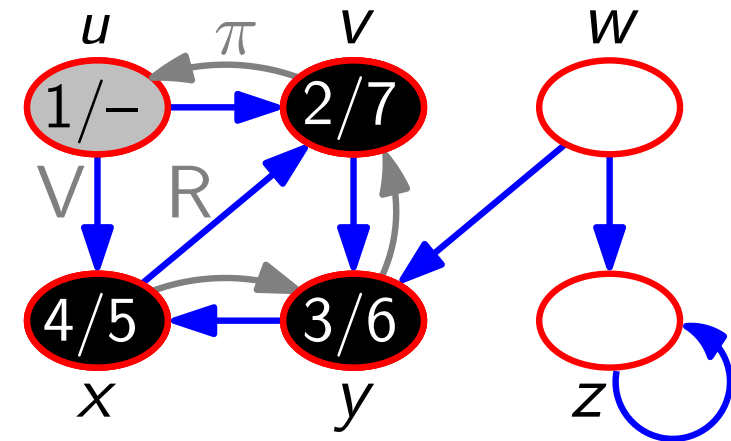
Eingabe: (un)gerichteter Graph  $G$

Ausgabe: – Besuchsintervalle ( $u.d/u.f$ )  
discovery time finish time

– DFS-Wald ( $\leftarrow \pi$ )

– Klassifizierung der Graphkanten:

- Baumkanten (Kanten von  $G_\pi$ )
- Rückwärtskanten (R)
- Vorwärtskanten (V)



*Farbe Zielknoten:*

*weiss*

*grau*

# Tiefensuche

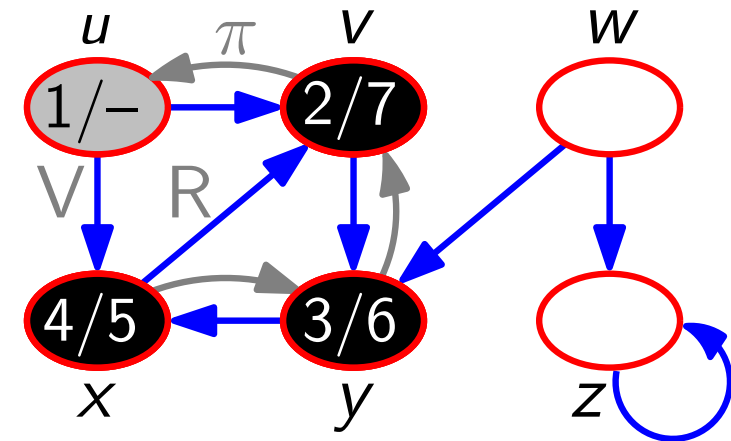
Eingabe: (un)gerichteter Graph  $G$

Ausgabe: – Besuchsintervalle ( $u.d/u.f$ )  
discovery time finish time

– DFS-Wald ( $\leftarrow \pi$ )

– Klassifizierung der Graphkanten:

- Baumkanten (Kanten von  $G_\pi$ )
- Rückwärtskanten (R)
- Vorwärtskanten (V)



*Farbe Zielknoten:*

*weiss*

*grau*

*schwarz*

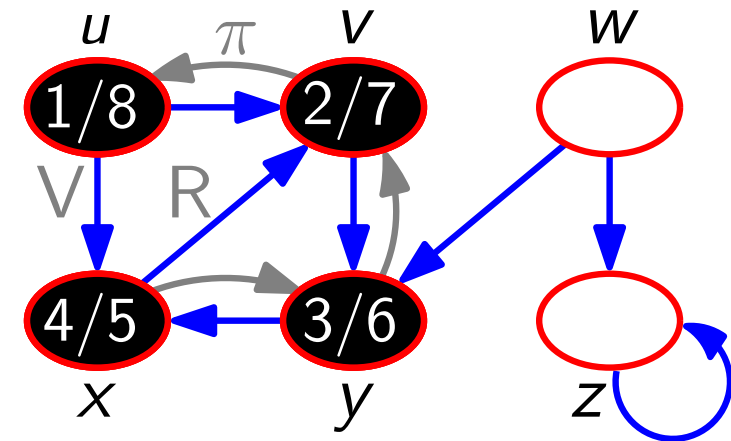
# Tiefensuche

Eingabe: (un)gerichteter Graph  $G$

Ausgabe: – Besuchsintervalle ( $u.d/u.f$ )  
 – DFS-Wald ( $\leftarrow \pi$ )

– Klassifizierung der Graphkanten:

- Baumkanten (Kanten von  $G_\pi$ )
- Rückwärtskanten (R)
- Vorwärtskanten (V)



*Farbe Zielknoten:*

*weiss*

*grau*

*schwarz*

# Tiefensuche

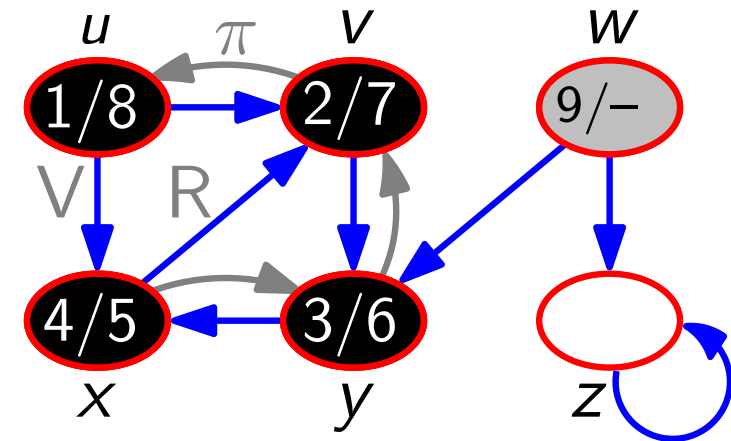
Eingabe: (un)gerichteter Graph  $G$

Ausgabe: – Besuchsintervalle ( $u.d/u.f$ )  
discovery time finish time

– DFS-Wald ( $\leftarrow \pi$ )

– Klassifizierung der Graphkanten:

- Baumkanten (Kanten von  $G_\pi$ )
- Rückwärtskanten (R)
- Vorwärtskanten (V)



*Farbe Zielknoten:*

*weiss*

*grau*

*schwarz*

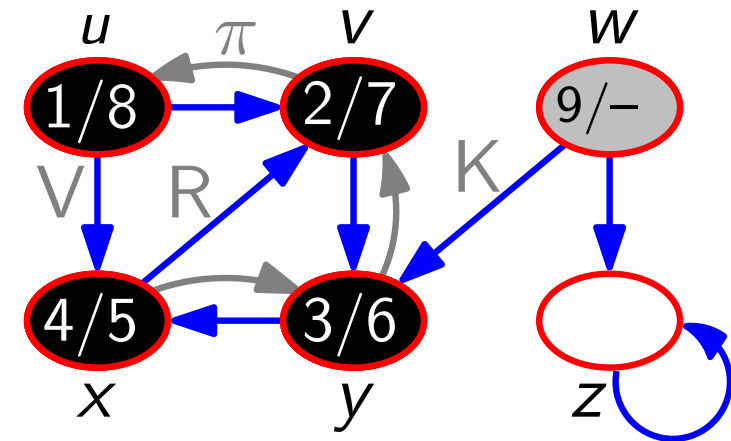
# Tiefensuche

Eingabe: (un)gerichteter Graph  $G$

Ausgabe: – Besuchsintervalle ( $u.d/u.f$ )  
 – DFS-Wald ( $\leftarrow \pi$ )

– Klassifizierung der Graphkanten:

- Baumkanten (Kanten von  $G_\pi$ )
- Rückwärtskanten (R)
- Vorwärtskanten (V)



*Farbe Zielknoten:*

*weiss*

*grau*

*schwarz*

# Tiefensuche

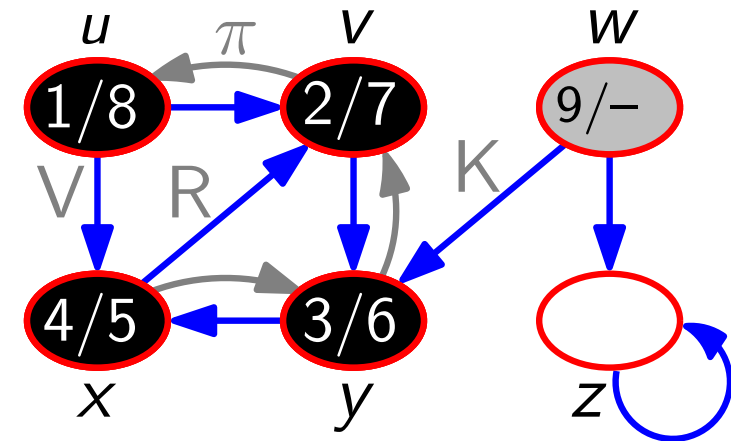
Eingabe: (un)gerichteter Graph  $G$

Ausgabe: – Besuchsintervalle ( $u.d/u.f$ )  
discovery time finish time

– DFS-Wald ( $\leftarrow \pi$ )

– Klassifizierung der Graphkanten:

- Baumkanten (Kanten von  $G_\pi$ )
- Rückwärtskanten (R)
- Vorwärtskanten (V)
- Kreuzkanten (K)



*Farbe Zielknoten:*

*weiss*

*grau*

*schwarz*

# Tiefensuche

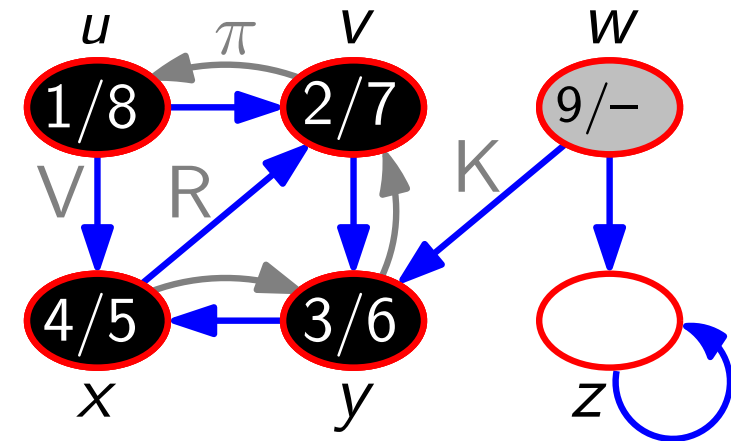
Eingabe: (un)gerichteter Graph  $G$

Ausgabe: – Besuchsintervalle ( $u.d/u.f$ )  
discovery time finish time

– DFS-Wald ( $\leftarrow \pi$ )

– Klassifizierung der Graphkanten:

- Baumkanten (Kanten von  $G_\pi$ )
- Rückwärtskanten (R)
- Vorwärtskanten (V)
- Kreuzkanten (K)



*Farbe Zielknoten:*

*weiss*

*grau*

*schwarz*

*schwarz*



# Tiefensuche

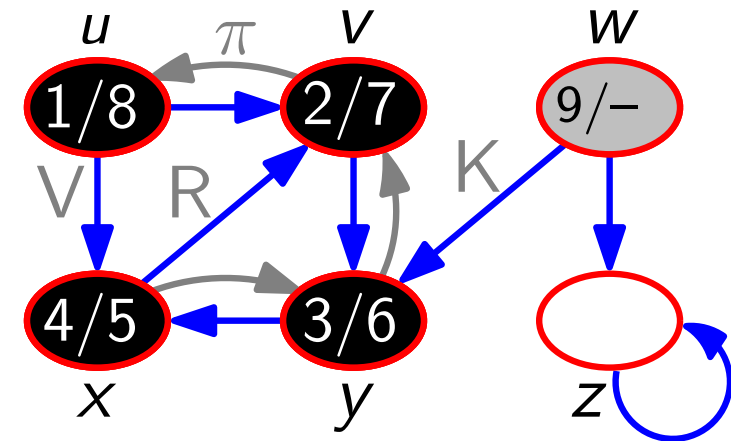
Eingabe: (un)gerichteter Graph  $G$

Ausgabe: – Besuchsintervalle ( $u.d/u.f$ )

– DFS-Wald ( $\leftarrow \pi$ )

– Klassifizierung der Graphkanten:

- Baumkanten (Kanten von  $G_\pi$ )
- Rückwärtskanten (R)
- Vorwärtskanten (V)
- Kreuzkanten (K)



*Farbe Zielknoten:*

*weiss*

*grau*

*schwarz und  
start.d < ziel.d*

*schwarz und  
start.d > ziel.d*

# Tiefensuche

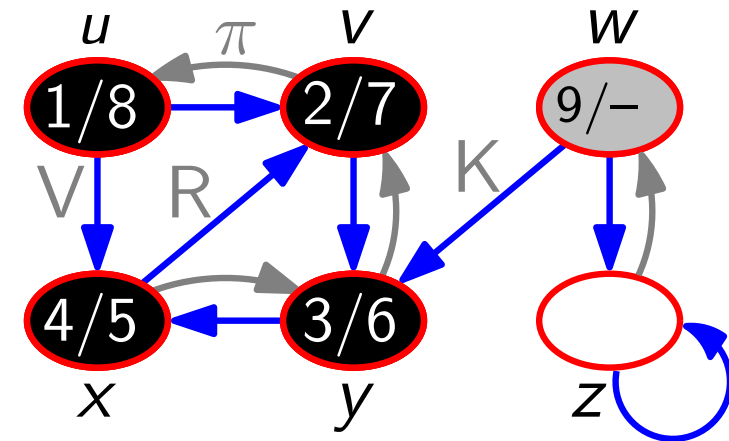
Eingabe: (un)gerichteter Graph  $G$

Ausgabe: – Besuchsintervalle ( $u.d/u.f$ )  
discovery time finish time

– DFS-Wald ( $\leftarrow \pi$ )

– Klassifizierung der Graphkanten:

- Baumkanten (Kanten von  $G_\pi$ )
- Rückwärtskanten (R)
- Vorwärtskanten (V)
- Kreuzkanten (K)



*Farbe Zielknoten:*

*weiss*

*grau*

*schwarz und  
start.d < ziel.d*

*schwarz und  
start.d > ziel.d*

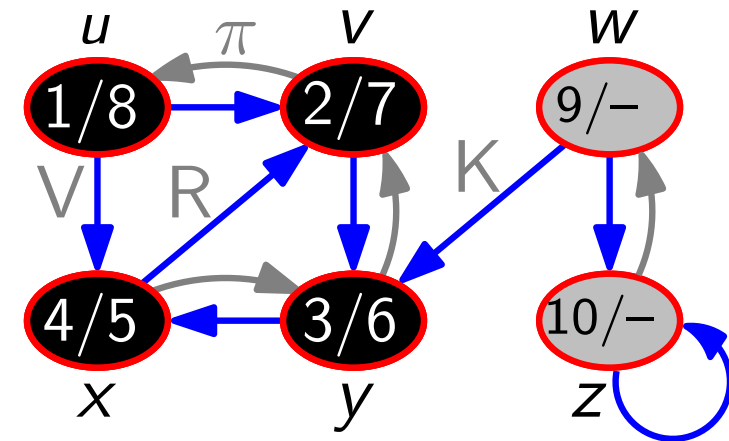
# Tiefensuche

Eingabe: (un)gerichteter Graph  $G$

Ausgabe: – Besuchsintervalle ( $u.d/u.f$ )  
 – DFS-Wald ( $\leftarrow \pi$ )

– Klassifizierung der Graphkanten:

- Baumkanten (Kanten von  $G_\pi$ )
- Rückwärtskanten (R)
- Vorwärtskanten (V)
- Kreuzkanten (K)



*Farbe Zielknoten:*

*weiss*

*grau*

*schwarz und  
start.d < ziel.d*

*schwarz und  
start.d > ziel.d*

# Tiefensuche

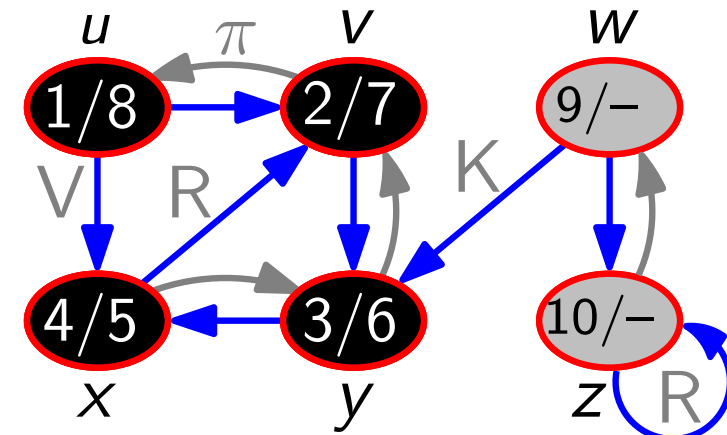
Eingabe: (un)gerichteter Graph  $G$

Ausgabe: – Besuchsintervalle ( $u.d/u.f$ )  
discovery time finish time

– DFS-Wald ( $\leftarrow \pi$ )

– Klassifizierung der Graphkanten:

- Baumkanten (Kanten von  $G_\pi$ )
- Rückwärtskanten (R)
- Vorwärtskanten (V)
- Kreuzkanten (K)



*Farbe Zielknoten:*

*weiss*

*grau*

*schwarz und  
start.d < ziel.d*

*schwarz und  
start.d > ziel.d*

# Tiefensuche

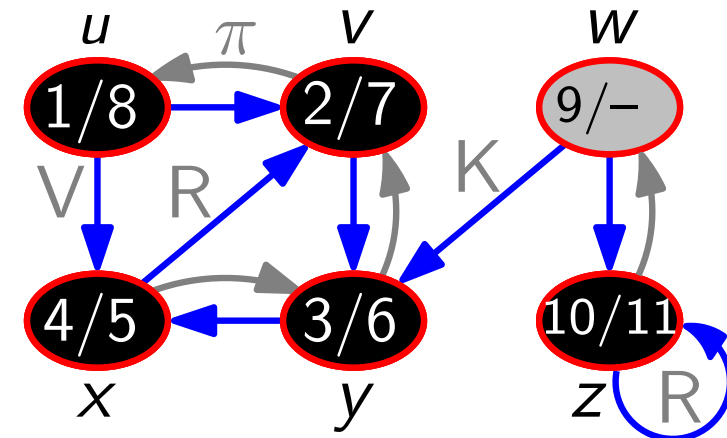
Eingabe: (un)gerichteter Graph  $G$

Ausgabe:

- Besuchsintervalle ( $u.d/u.f$ )  
discovery time finish time
- DFS-Wald ( $\leftarrow \pi$ )

– Klassifizierung der Graphkanten:

- Baumkanten (Kanten von  $G_\pi$ )
- Rückwärtskanten (R)
- Vorwärtskanten (V)
- Kreuzkanten (K)



*Farbe Zielknoten:*

*weiss*

*grau*

*schwarz und  
start.d < ziel.d*

*schwarz und  
start.d > ziel.d*

# Tiefensuche

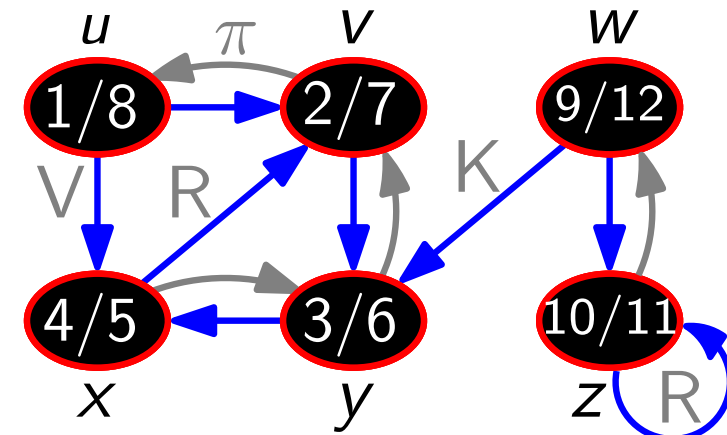
Eingabe: (un)gerichteter Graph  $G$

Ausgabe: – Besuchsintervalle ( $u.d/u.f$ )  
discovery time finish time

– DFS-Wald ( $\leftarrow \pi$ )

– Klassifizierung der Graphkanten:

- Baumkanten (Kanten von  $G_\pi$ )
- Rückwärtskanten (R)
- Vorwärtskanten (V)
- Kreuzkanten (K)



*Farbe Zielknoten:*

*weiss*

*grau*

*schwarz und  
start.d < ziel.d*

*schwarz und  
start.d > ziel.d*

# Tiefensuche

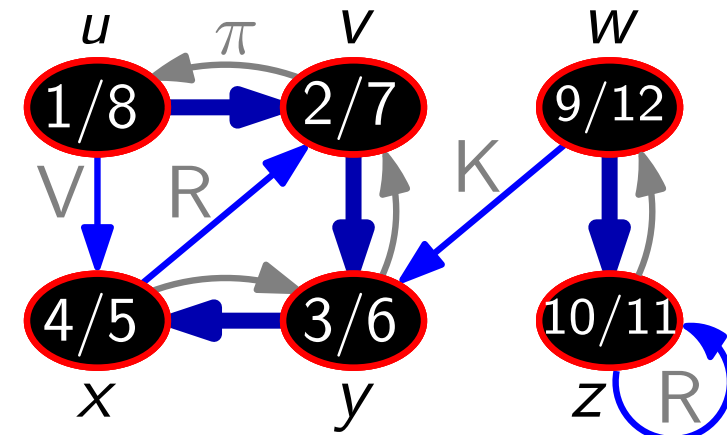
Eingabe: (un)gerichteter Graph  $G$

Ausgabe: – Besuchsintervalle ( $u.d/u.f$ )  
discovery time finish time

– DFS-Wald ( $\leftarrow \pi$ )

– Klassifizierung der Graphkanten:

- Baumkanten (Kanten von  $G_\pi$ )
- Rückwärtskanten (R)
- Vorwärtskanten (V)
- Kreuzkanten (K)



*Farbe Zielknoten:*

*weiss*

*grau*

*schwarz und  
start.d < ziel.d*

*schwarz und  
start.d > ziel.d*

# Tiefensuche

Eingabe: (un)gerichteter Graph  $G$

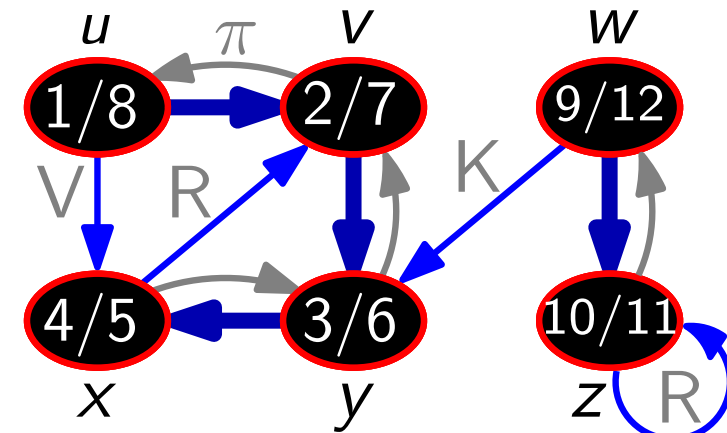
Ausgabe: – Besuchsintervalle ( $u.d/u.f$ )  
 – DFS-Wald ( $\leftarrow \pi$ )

– Klassifizierung der Graphkanten:

- Baumkanten (Kanten von  $G_\pi$ )

Kanten des DFS-Waldes (entgegen  $\pi$  gerichtet)

- Rückwärtskanten (R)
- Vorwärtskanten (V)
- Kreuzkanten (K)



Farbe Zielknoten:

*weiss*

*grau*

*schwarz und*  
 $start.d < ziel.d$

*schwarz und*  
 $start.d > ziel.d$



# Tiefensuche

Eingabe: (un)gerichteter Graph  $G$

Ausgabe:

- Besuchsintervalle ( $u.d/u.f$ )  
discovery time finish time
- DFS-Wald ( $\leftarrow \pi$ )
- Klassifizierung der Graphkanten:

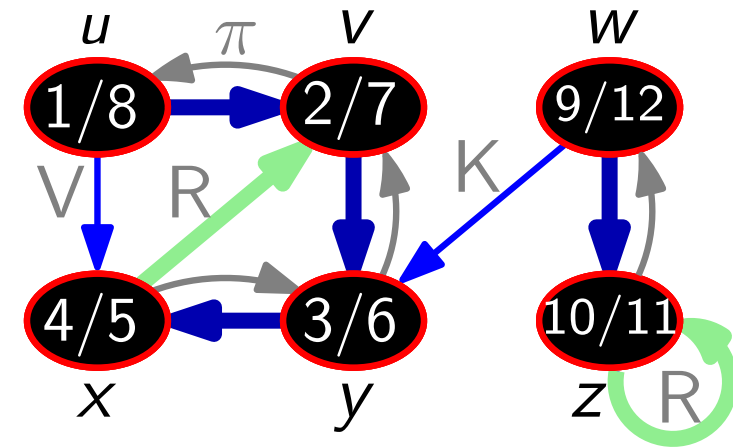
- Baumkanten (Kanten von  $G_\pi$ )

Kanten des DFS-Waldes (entgegen  $\pi$  gerichtet)

- Rückwärtskanten (R)

- Vorwärtskanten (V)

- Kreuzkanten (K)



*Farbe Zielknoten:*

*weiss*

*grau*

*schwarz und*  
*start.d < ziel.d*

*schwarz und*  
*start.d > ziel.d*

# Tiefensuche

Eingabe: (un)gerichteter Graph  $G$

Ausgabe:

- Besuchsintervalle ( $u.d/u.f$ )  
discovery time    finish time
- DFS-Wald ( $\leftarrow \pi$ )
- Klassifizierung der Graphkanten:

- Baumkanten (Kanten von  $G_\pi$ )

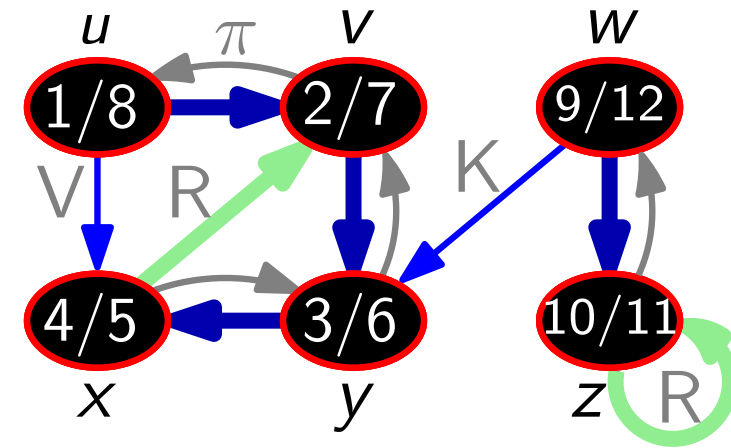
Kanten des DFS-Waldes (entgegen  $\pi$  gerichtet)

- Rückwärtskanten (R)

Nicht-Baumkanten zu einem Vorgängerknoten

- Vorwärtskanten (V)

- Kreuzkanten (K)



*Farbe Zielknoten:*

*weiss*

*grau*

*schwarz und*  
*start.d < ziel.d*

*schwarz und*  
*start.d > ziel.d*

# Tiefensuche

Eingabe: (un)gerichteter Graph  $G$

Ausgabe:

- Besuchsintervalle ( $u.d/u.f$ )  
discovery time    finish time
- DFS-Wald ( $\leftarrow \pi$ )
- Klassifizierung der Graphkanten:

- Baumkanten (Kanten von  $G_\pi$ )

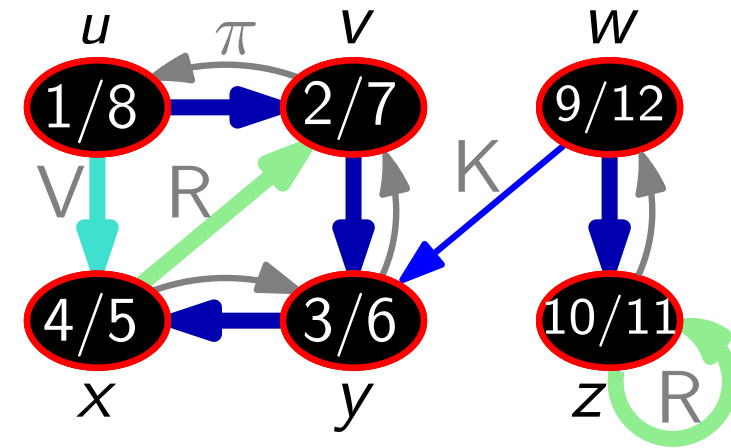
Kanten des DFS-Waldes (entgegen  $\pi$  gerichtet)

- Rückwärtskanten (R)

Nicht-Baumkanten zu einem Vorgängerknoten

- Vorwärtskanten (V)

- Kreuzkanten (K)



*Farbe Zielknoten:*

*weiss*

*grau*

*schwarz und  
start.d < ziel.d*

*schwarz und  
start.d > ziel.d*

# Tiefensuche

Eingabe: (un)gerichteter Graph  $G$

Ausgabe: – Besuchsintervalle ( $u.d/u.f$ )  
 – DFS-Wald ( $\leftarrow \pi$ )  
 – Klassifizierung der Graphkanten:

● Baumkanten (Kanten von  $G_\pi$ )

Kanten des DFS-Waldes (entgegen  $\pi$  gerichtet)

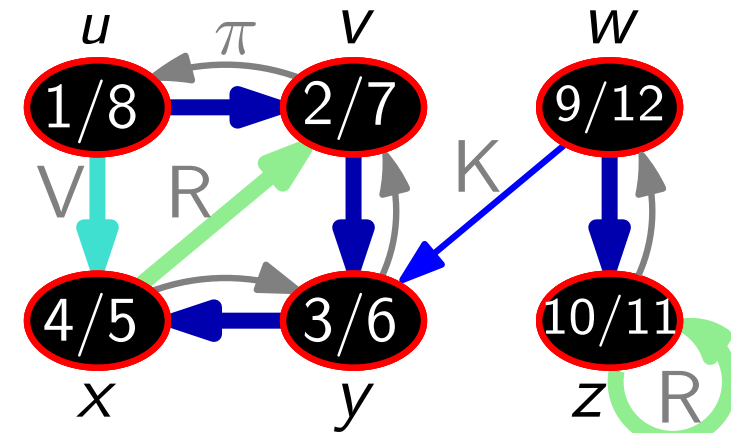
● Rückwärtskanten (R)

Nicht-Baumkanten zu einem Vorgängerknoten

● Vorwärtskanten (V)

Nicht-Baumkanten zu einem Nachfolgerknoten

● Kreuzkanten (K)



*Farbe Zielknoten:*

*weiss*

*grau*

*schwarz und  
start.d < ziel.d*

*schwarz und  
start.d > ziel.d*

# Tiefensuche

Eingabe: (un)gerichteter Graph  $G$

Ausgabe:

- Besuchsintervalle ( $u.d/u.f$ )  
discovery time finish time
- DFS-Wald ( $\leftarrow \pi$ )
- Klassifizierung der Graphkanten:

- Baumkanten (Kanten von  $G_\pi$ )

Kanten des DFS-Waldes (entgegen  $\pi$  gerichtet)

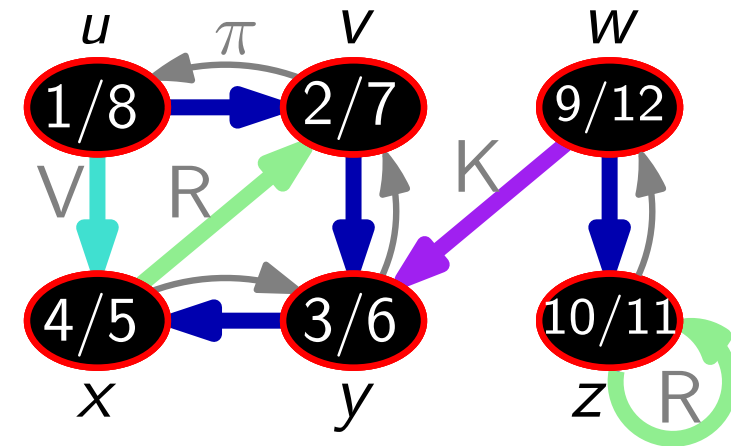
- Rückwärtskanten (R)

Nicht-Baumkanten zu einem Vorgängerknoten

- Vorwärtskanten (V)

Nicht-Baumkanten zu einem Nachfolgerknoten

- Kreuzkanten (K)



*Farbe Zielknoten:*

*weiss*

*grau*

*schwarz und*  
*start.d < ziel.d*

*schwarz und*  
*start.d > ziel.d*

# Tiefensuche

Eingabe: (un)gerichteter Graph  $G$

Ausgabe:

- Besuchsintervalle ( $u.d/u.f$ )  
discovery time finish time
- DFS-Wald ( $\leftarrow \pi$ )
- Klassifizierung der Graphkanten:

- Baumkanten (Kanten von  $G_\pi$ )

Kanten des DFS-Waldes (entgegen  $\pi$  gerichtet)

- Rückwärtskanten (R)

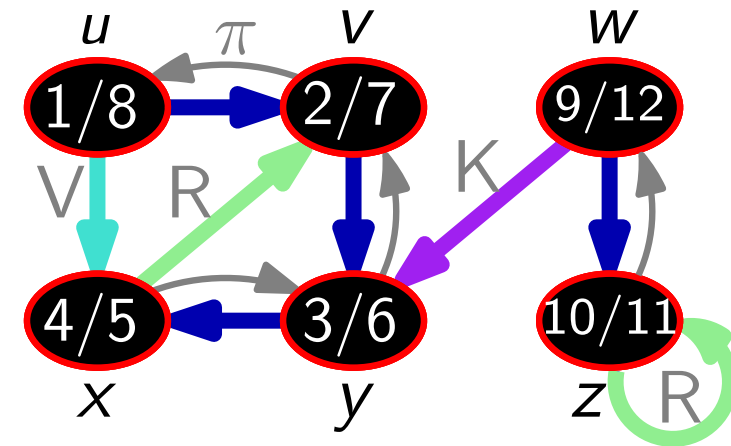
Nicht-Baumkanten zu einem Vorgängerknoten

- Vorwärtskanten (V)

Nicht-Baumkanten zu einem Nachfolgerknoten

- Kreuzkanten (K)

Kanten, bei denen kein Endpunkt Vorgänger des anderen ist.



*Farbe Zielknoten:*

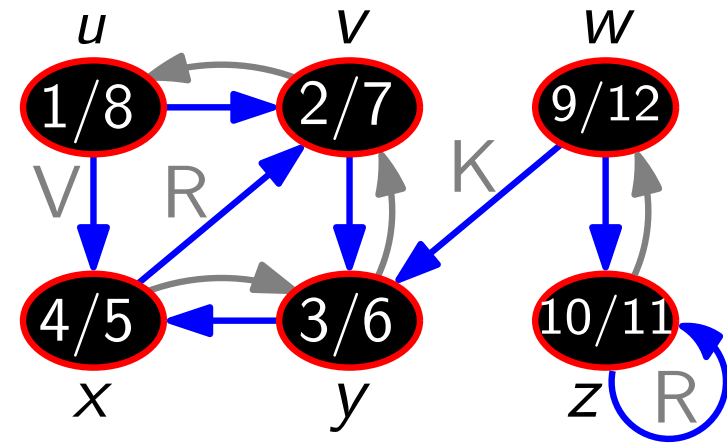
*weiss*

*grau*

*schwarz und  
start.d < ziel.d*

*schwarz und  
start.d > ziel.d*

# Tiefensuche – Pseudocode



# Tiefensuche – Pseudocode

```
DFS(Graph  $G = (V, E)$ )
```

```
  foreach  $u \in V$  do
```

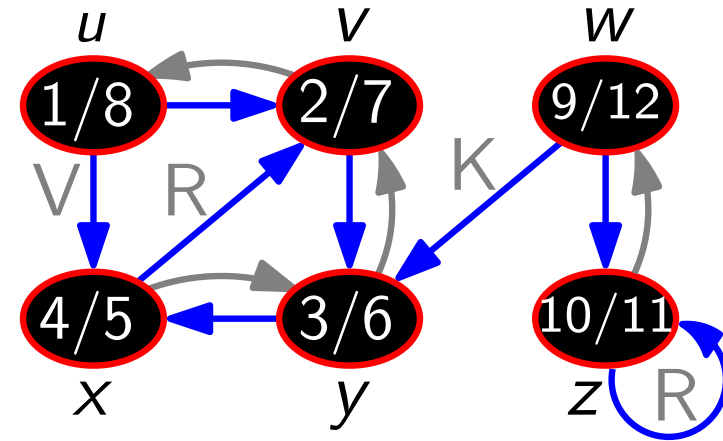
```
     $u.color = white$ 
```

```
     $u.\pi = nil$ 
```

```
   $time = 0$  // globale Variable!
```

```
  foreach  $u \in V$  do
```

```
    if  $u.color == white$  then DFSVisit( $G, u$ )
```





# Tiefensuche – Pseudocode

```
DFS(Graph  $G = (V, E)$ )
```

```
  foreach  $u \in V$  do
```

```
     $u.color = white$ 
```

```
     $u.\pi = nil$ 
```

```
   $time = 0$  // globale Variable!
```

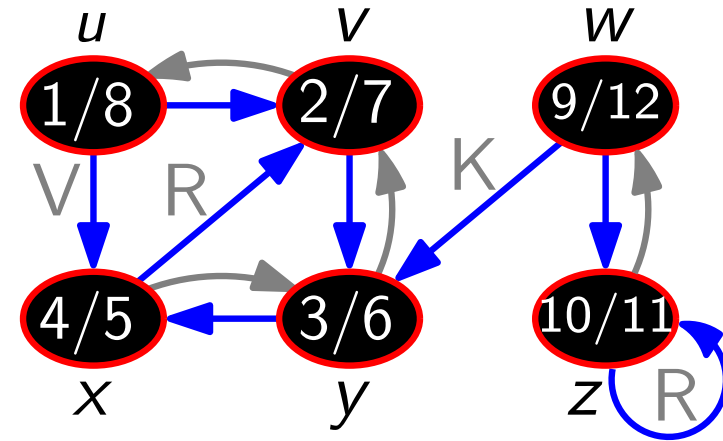
```
  foreach  $u \in V$  do
```

```
    if  $u.color == white$  then DFSVisit( $G, u$ )
```

```
DFSVisit(Graph  $G$ , Vertex  $u$ )
```

```
   $time = time + 1$ 
```

```
   $u.d = time; u.color = gray$ 
```



# Tiefensuche – Pseudocode

```
DFS(Graph  $G = (V, E)$ )
```

```
  foreach  $u \in V$  do
```

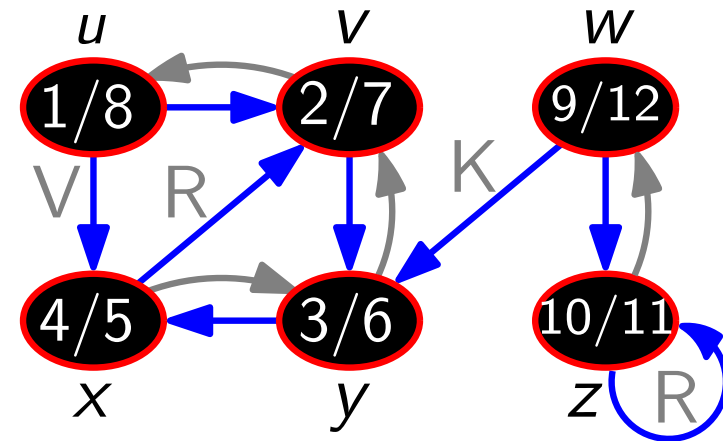
```
     $u.color = white$ 
```

```
     $u.\pi = nil$ 
```

```
   $time = 0$  // globale Variable!
```

```
  foreach  $u \in V$  do
```

```
    if  $u.color == white$  then DFSVisit( $G, u$ )
```



```
DFSVisit(Graph  $G$ , Vertex  $u$ )
```

```
   $time = time + 1$ 
```

```
   $u.d = time$ ;  $u.color = gray$ 
```

Für jeden Knoten  $u$  von  $G$  ist

- $u.d$  der Zeitpunkt der Entdeckung,
  - $u.f$  der Abschluss-Zeitpunkt;
- Besuchsintervall von  $u$  ist  $[u.d, u.f]$ .

# Tiefensuche – Pseudocode

```
DFS(Graph  $G = (V, E)$ )
```

```
  foreach  $u \in V$  do
```

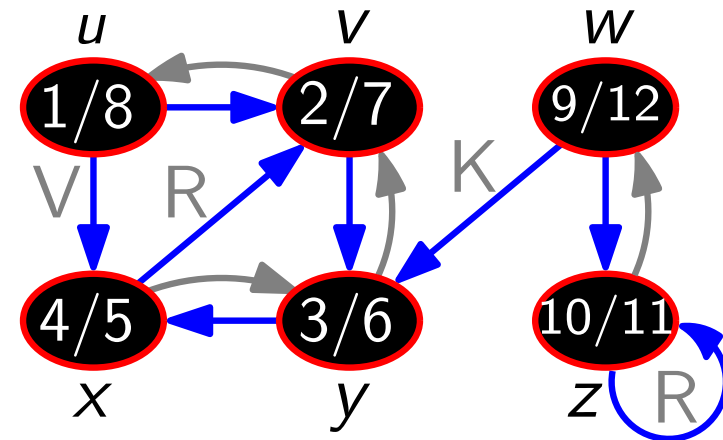
```
     $u.color = white$ 
```

```
     $u.\pi = nil$ 
```

```
   $time = 0$  // globale Variable!
```

```
  foreach  $u \in V$  do
```

```
    if  $u.color == white$  then DFSVisit( $G, u$ )
```



```
DFSVisit(Graph  $G$ , Vertex  $u$ )
```

```
   $time = time + 1$ 
```

```
   $u.d = time$ ;  $u.color = gray$ 
```

```
  foreach  $v \in Adj[u]$  do
```

```
    |
```

Für jeden Knoten  $u$  von  $G$  ist

- $u.d$  der Zeitpunkt der Entdeckung,
  - $u.f$  der Abschluss-Zeitpunkt;
- Besuchsintervall von  $u$  ist  $[u.d, u.f]$ .

# Tiefensuche – Pseudocode

```
DFS(Graph  $G = (V, E)$ )
```

```
  foreach  $u \in V$  do
```

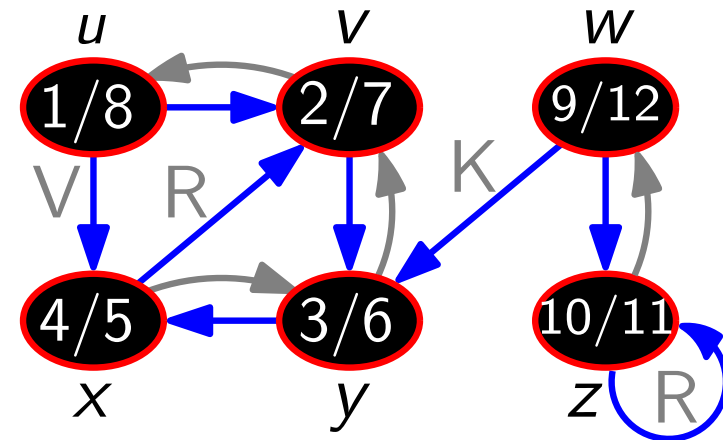
```
     $u.color = white$ 
```

```
     $u.\pi = nil$ 
```

```
   $time = 0$  // globale Variable!
```

```
  foreach  $u \in V$  do
```

```
    if  $u.color == white$  then DFSVisit( $G, u$ )
```



```
DFSVisit(Graph  $G$ , Vertex  $u$ )
```

```
   $time = time + 1$ 
```

```
   $u.d = time$ ;  $u.color = gray$ 
```

```
  foreach  $v \in Adj[u]$  do
```

*Ergänzen Sie den Code  
in und nach der foreach-Schleife!  
Benutzen Sie Rekursion.*

Für jeden Knoten  $u$  von  $G$  ist

- $u.d$  der Zeitpunkt der Entdeckung,
- $u.f$  der Abschluss-Zeitpunkt;

Besuchsintervall von  $u$  ist  $[u.d, u.f]$ .

# Tiefensuche – Pseudocode

```
DFS(Graph  $G = (V, E)$ )
```

```
  foreach  $u \in V$  do
```

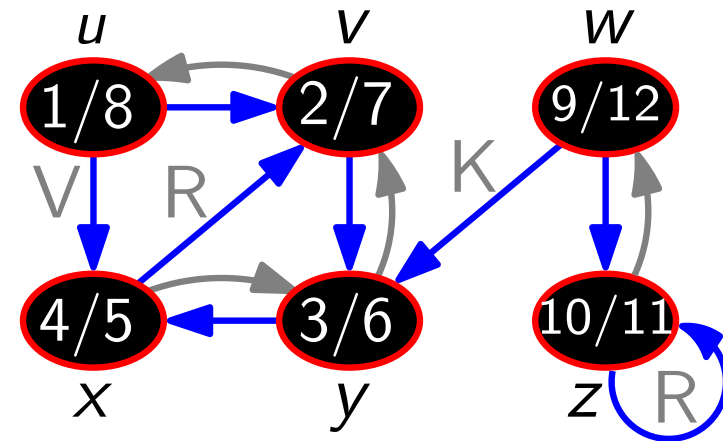
```
     $u.color = white$ 
```

```
     $u.\pi = nil$ 
```

```
   $time = 0$  // globale Variable!
```

```
  foreach  $u \in V$  do
```

```
    if  $u.color == white$  then DFSVisit( $G, u$ )
```



```
DFSVisit(Graph  $G$ , Vertex  $u$ )
```

```
   $time = time + 1$ 
```

```
   $u.d = time$ ;  $u.color = gray$ 
```

```
  foreach  $v \in Adj[u]$  do
```

```
    if  $v.color == white$  then
```

```
       $v.\pi = u$ ; DFSVisit( $G, v$ )
```

```
   $time = time + 1$ 
```

```
   $u.f = time$ ;  $u.color = black$ 
```

Für jeden Knoten  $u$  von  $G$  ist

- $u.d$  der Zeitpunkt der Entdeckung,
  - $u.f$  der Abschluss-Zeitpunkt;
- Besuchsintervall von  $u$  ist  $[u.d, u.f]$ .

# Tiefensuche – Pseudocode

```
DFS(Graph  $G = (V, E)$ )
```

```
  foreach  $u \in V$  do
```

```
     $u.color = white$ 
```

```
     $u.\pi = nil$ 
```

```
   $time = 0$  // globale Variable!
```

```
  foreach  $u \in V$  do
```

```
    if  $u.color == white$  then DFSVisit( $G, u$ )
```

```
DFSVisit(Graph  $G$ , Vertex  $u$ )
```

```
   $time = time + 1$ 
```

```
   $u.d = time$ ;  $u.color = gray$ 
```

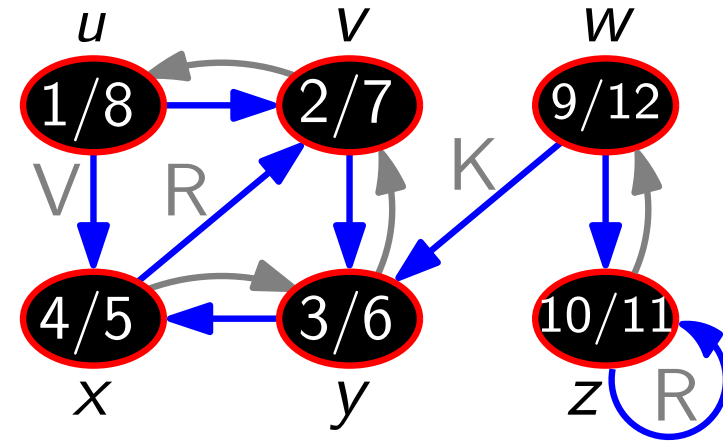
```
  foreach  $v \in Adj[u]$  do
```

```
    if  $v.color == white$  then
```

```
       $v.\pi = u$ ; DFSVisit( $G, v$ )
```

```
   $time = time + 1$ 
```

```
   $u.f = time$ ;  $u.color = black$ 
```



# Tiefensuche – Pseudocode

```
DFS(Graph  $G = (V, E)$ )
```

```
  foreach  $u \in V$  do
```

```
     $u.color = white$ 
```

```
     $u.\pi = nil$ 
```

```
   $time = 0$  // globale Variable!
```

```
  foreach  $u \in V$  do
```

```
    if  $u.color == white$  then DFSVisit( $G, u$ )
```

```
DFSVisit(Graph  $G$ , Vertex  $u$ )
```

```
   $time = time + 1$ 
```

```
   $u.d = time$ ;  $u.color = gray$ 
```

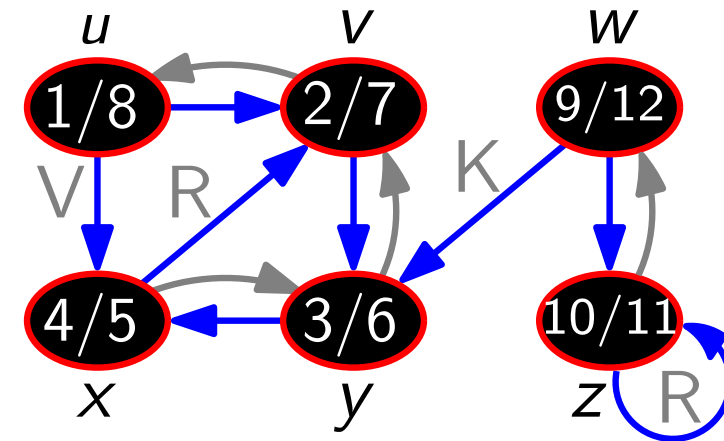
```
  foreach  $v \in Adj[u]$  do
```

```
    if  $v.color == white$  then
```

```
       $v.\pi = u$ ; DFSVisit( $G, v$ )
```

```
   $time = time + 1$ 
```

```
   $u.f = time$ ;  $u.color = black$ 
```



**Laufzeit  
von DFS?**

# Tiefensuche – Pseudocode

```
DFS(Graph  $G = (V, E)$ )
```

```
  foreach  $u \in V$  do
```

```
     $u.color = white$ 
```

```
     $u.\pi = nil$ 
```

```
   $time = 0$  // globale Variable!
```

```
  foreach  $u \in V$  do
```

```
    if  $u.color == white$  then DFSVisit( $G, u$ )
```

```
DFSVisit(Graph  $G$ , Vertex  $u$ )
```

```
   $time = time + 1$ 
```

```
   $u.d = time$ ;  $u.color = gray$ 
```

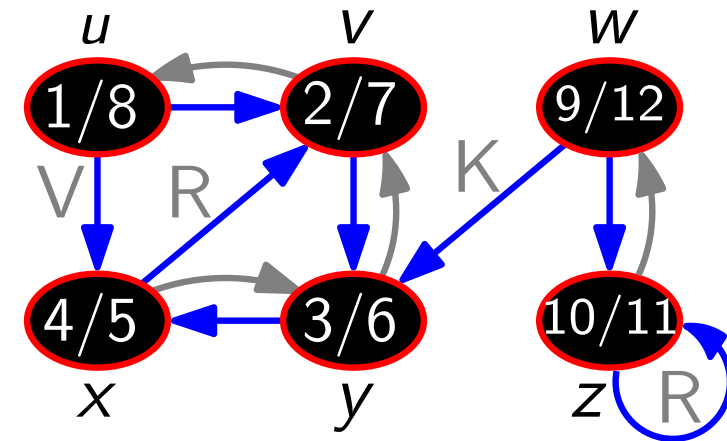
```
  foreach  $v \in Adj[u]$  do
```

```
    if  $v.color == white$  then
```

```
       $v.\pi = u$ ; DFSVisit( $G, v$ )
```

```
   $time = time + 1$ 
```

```
   $u.f = time$ ;  $u.color = black$ 
```



**Laufzeit  
von DFS?**

- DFSVisit wird nur für weiße Knoten aufgerufen.



# Tiefensuche – Pseudocode

```
DFS(Graph  $G = (V, E)$ )
```

```
  foreach  $u \in V$  do
```

```
     $u.color = white$ 
```

```
     $u.\pi = nil$ 
```

```
   $time = 0$  // globale Variable!
```

```
  foreach  $u \in V$  do
```

```
    if  $u.color == white$  then DFSVisit( $G, u$ )
```

```
DFSVisit(Graph  $G$ , Vertex  $u$ )
```

```
   $time = time + 1$ 
```

```
   $u.d = time$ ;  $u.color = gray$ 
```

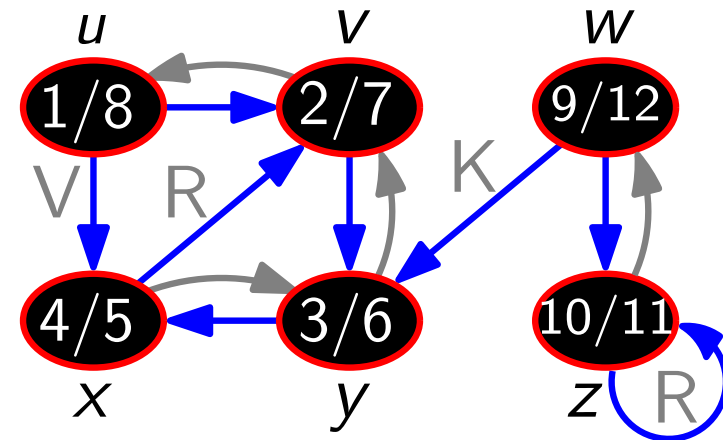
```
  foreach  $v \in Adj[u]$  do
```

```
    if  $v.color == white$  then
```

```
       $v.\pi = u$ ; DFSVisit( $G, v$ )
```

```
   $time = time + 1$ 
```

```
   $u.f = time$ ;  $u.color = black$ 
```



## Laufzeit von DFS?

- DFSVisit wird nur für weiße Knoten aufgerufen.
- In DFSVisit wird der neue Knoten sofort grau gefärbt.

# Tiefensuche – Pseudocode

```
DFS(Graph  $G = (V, E)$ )
```

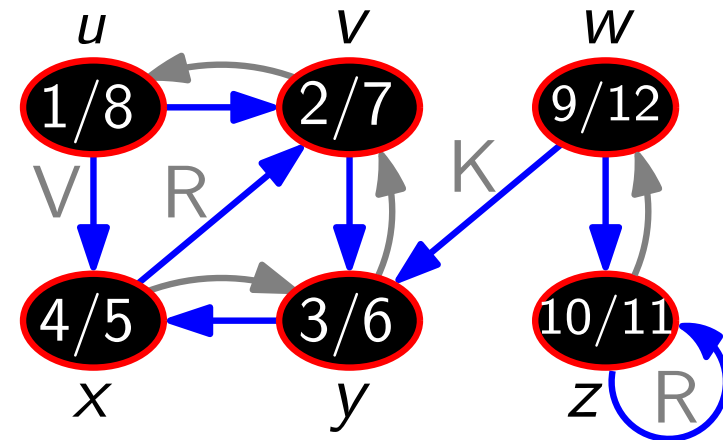
```
  foreach  $u \in V$  do
```

```
     $u.color = white$   
     $u.\pi = nil$ 
```

```
   $time = 0$  // globale Variable!
```

```
  foreach  $u \in V$  do
```

```
    if  $u.color == white$  then DFSVisit( $G, u$ )
```



## Laufzeit von DFS?

```
DFSVisit(Graph  $G$ , Vertex  $u$ )
```

```
   $time = time + 1$ 
```

```
   $u.d = time$ ;  $u.color = gray$ 
```

```
  foreach  $v \in Adj[u]$  do
```

```
    if  $v.color == white$  then  
       $v.\pi = u$ ; DFSVisit( $G, v$ )
```

```
   $time = time + 1$ 
```

```
   $u.f = time$ ;  $u.color = black$ 
```

- DFSVisit wird nur für weiße Knoten aufgerufen.
  - In DFSVisit wird der neue Knoten sofort grau gefärbt.
- ⇒ DFSVisit wird für jeden Knoten genau 1× aufgerufen.

# Tiefensuche – Pseudocode

```
DFS(Graph  $G = (V, E)$ )
```

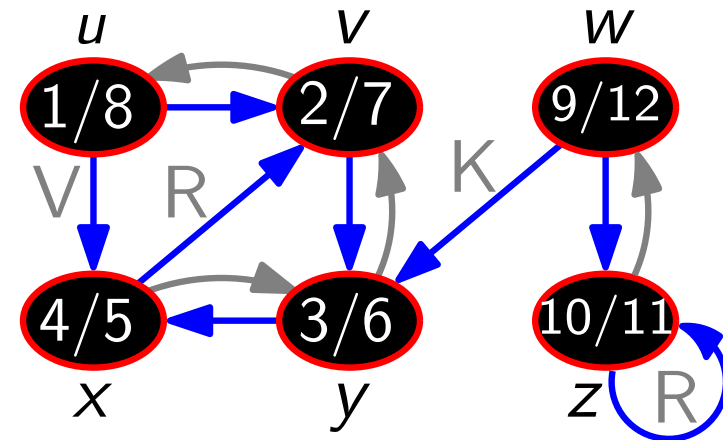
```
  foreach  $u \in V$  do
```

```
     $u.color = white$   
     $u.\pi = nil$ 
```

```
   $time = 0$  // globale Variable!
```

```
  foreach  $u \in V$  do
```

```
    if  $u.color == white$  then DFSVisit( $G, u$ )
```



**Laufzeit  
von DFS?**

```
DFSVisit(Graph  $G$ , Vertex  $u$ )
```

```
   $time = time + 1$ 
```

```
   $u.d = time$ ;  $u.color = gray$ 
```

```
  foreach  $v \in Adj[u]$  do
```

```
    if  $v.color == white$  then  
       $v.\pi = u$ ; DFSVisit( $G, v$ )
```

```
   $time = time + 1$ 
```

```
   $u.f = time$ ;  $u.color = black$ 
```

- DFSVisit wird nur für weiße Knoten aufgerufen.
- In DFSVisit wird der neue Knoten sofort grau gefärbt.  
⇒ DFSVisit wird für jeden Knoten genau 1× aufgerufen.
- DFS ohne if  $O(V)$  Zeit  
DFSVisit ohne Rek.  $O(\deg^{(out)} u)$

# Tiefensuche – Pseudocode

```
DFS(Graph  $G = (V, E)$ )
```

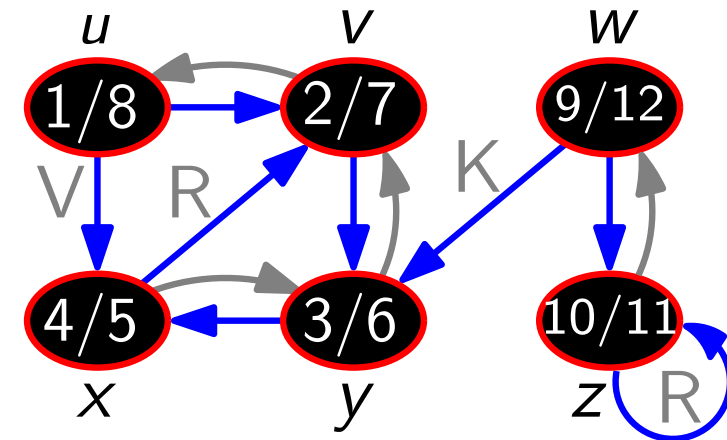
```
  foreach  $u \in V$  do
```

```
     $u.color = white$   
     $u.\pi = nil$ 
```

```
   $time = 0$  // globale Variable!
```

```
  foreach  $u \in V$  do
```

```
    if  $u.color == white$  then DFSVisit( $G, u$ )
```



**Laufzeit  
von DFS?**

```
DFSVisit(Graph  $G$ , Vertex  $u$ )
```

```
   $time = time + 1$ 
```

```
   $u.d = time$ ;  $u.color = gray$ 
```

```
  foreach  $v \in Adj[u]$  do
```

```
    if  $v.color == white$  then  
       $v.\pi = u$ ; DFSVisit( $G, v$ )
```

```
   $time = time + 1$ 
```

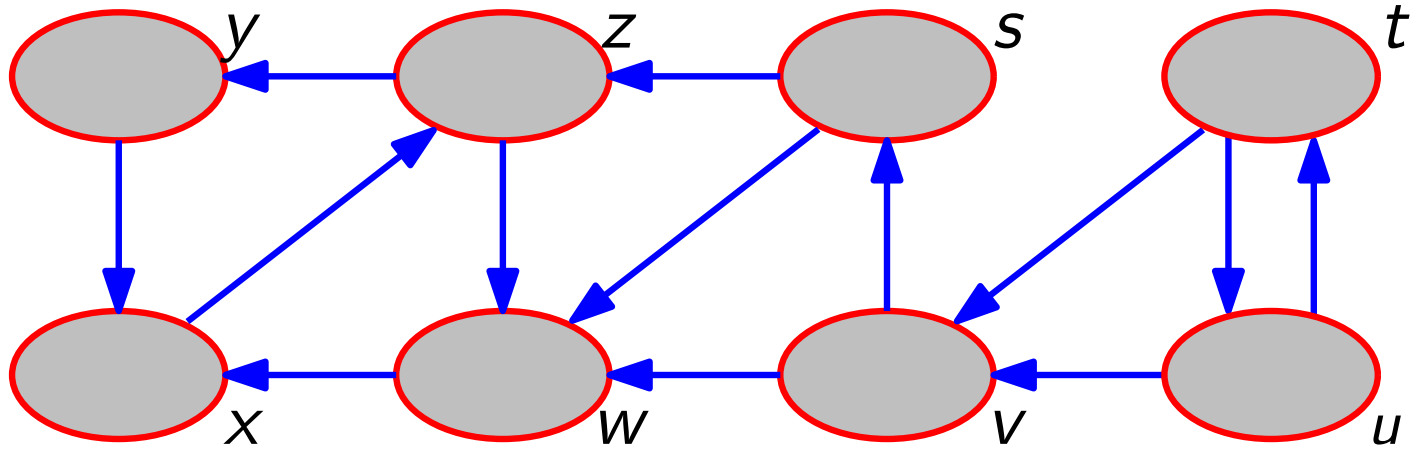
```
   $u.f = time$ ;  $u.color = black$ 
```

- DFSVisit wird nur für weiße Knoten aufgerufen.
- In DFSVisit wird der neue Knoten sofort grau gefärbt.  
⇒ DFSVisit wird für jeden Knoten genau 1× aufgerufen.
- DFS ohne if  $O(V)$  Zeit  
DFSVisit ohne Rek.  $O(\deg u)$   

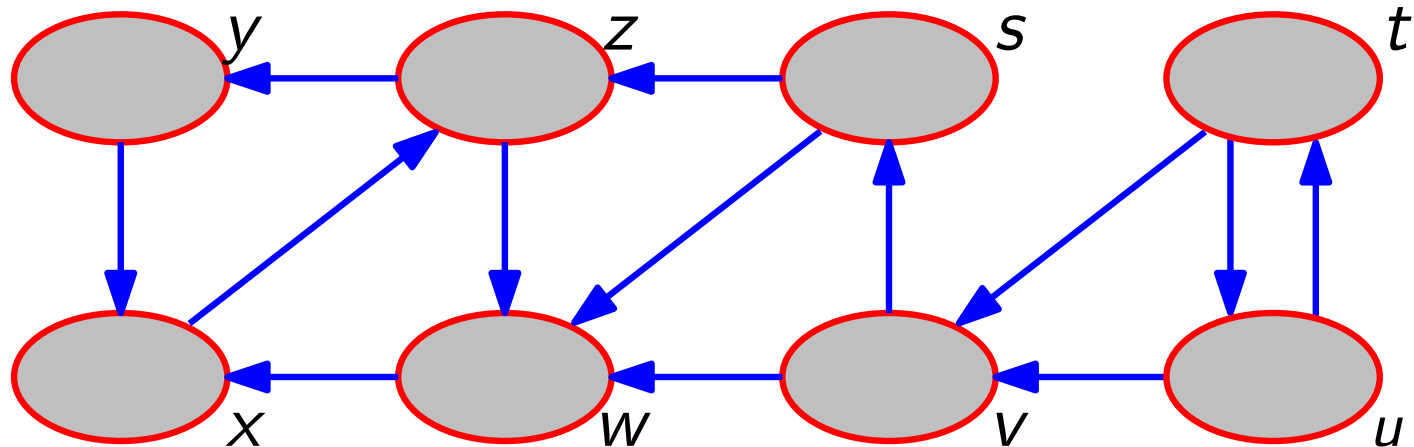
---

DFS gesamt  $O(V + E)$  Zeit

# Tiefensuche – Eigenschaften



# Tiefensuche – Eigenschaften

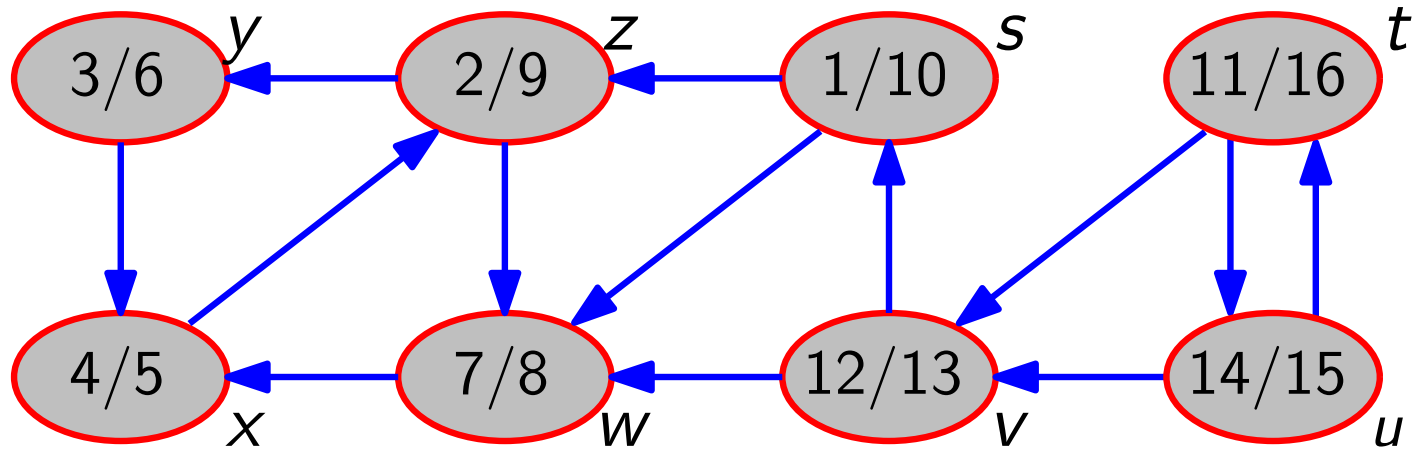


**Aufgabe:** Kopieren Sie obigen Graphen.

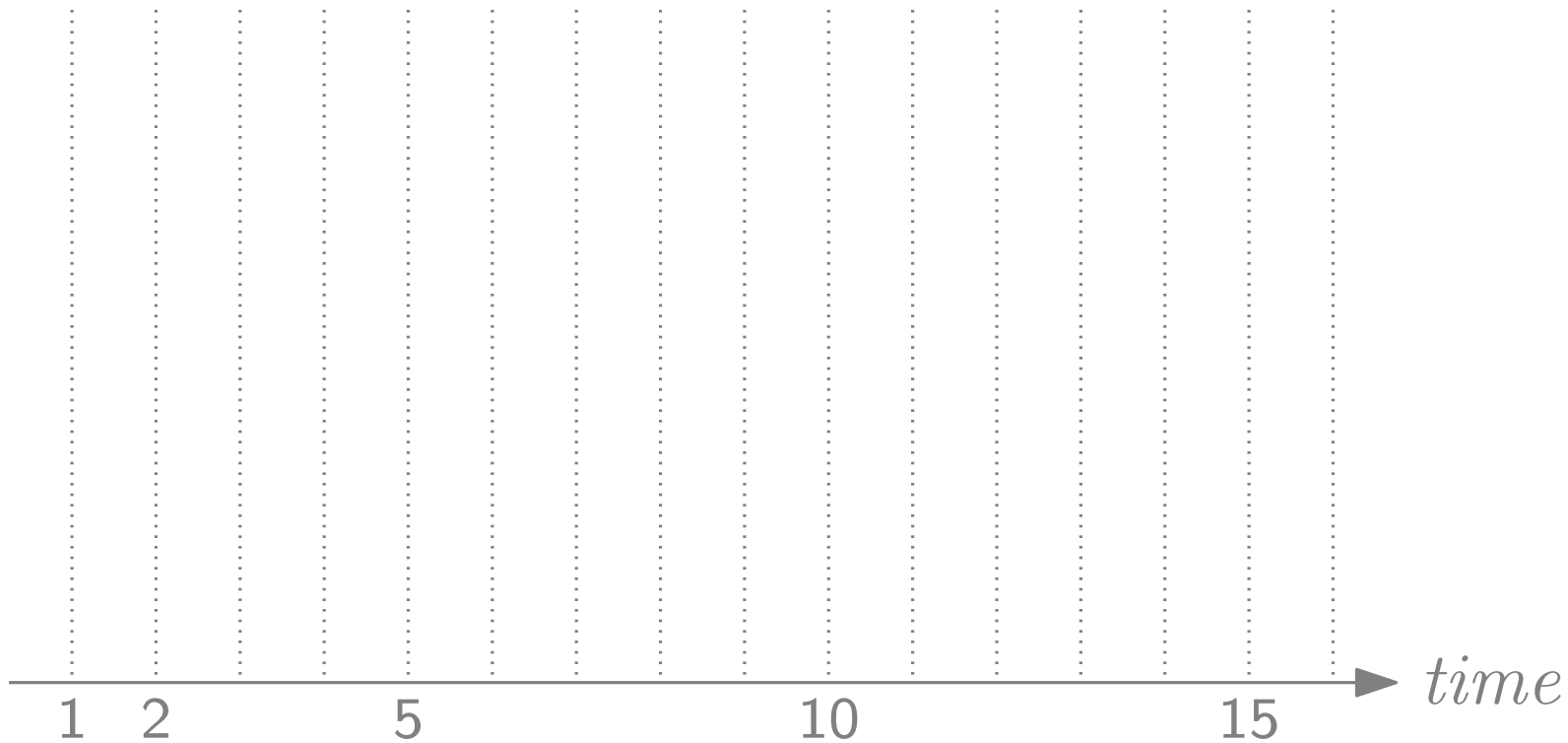
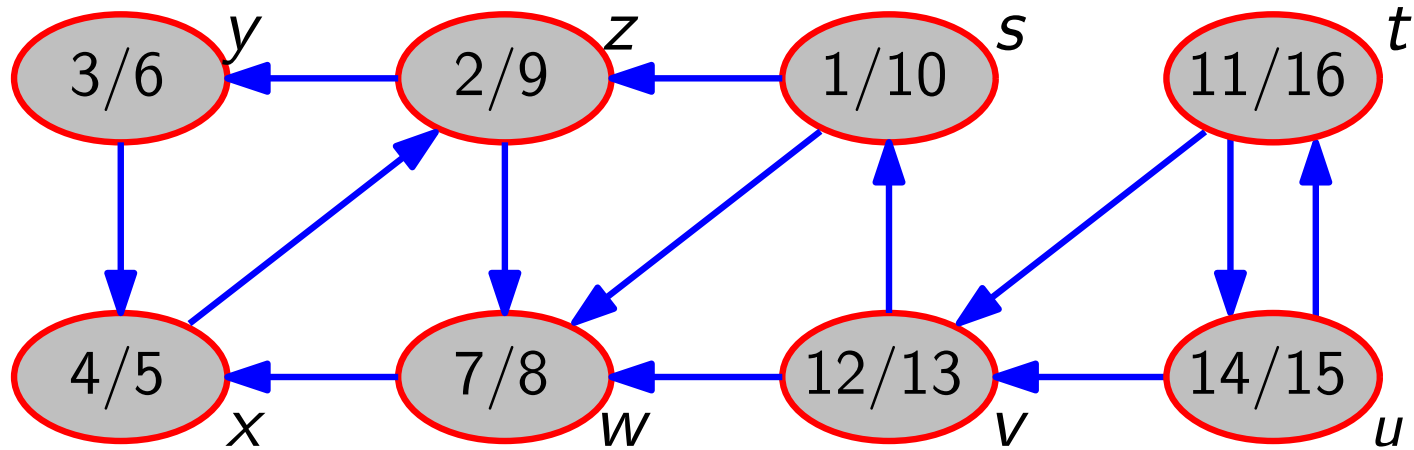
Berechnen Sie dann mit DFS alle Besuchsintervalle.

Beginnen Sie mit  $s$ . Wenn Sie eine Wahl haben, nehmen Sie zuerst den *obersten* verfügbaren Knoten.

# Tiefensuche – Eigenschaften

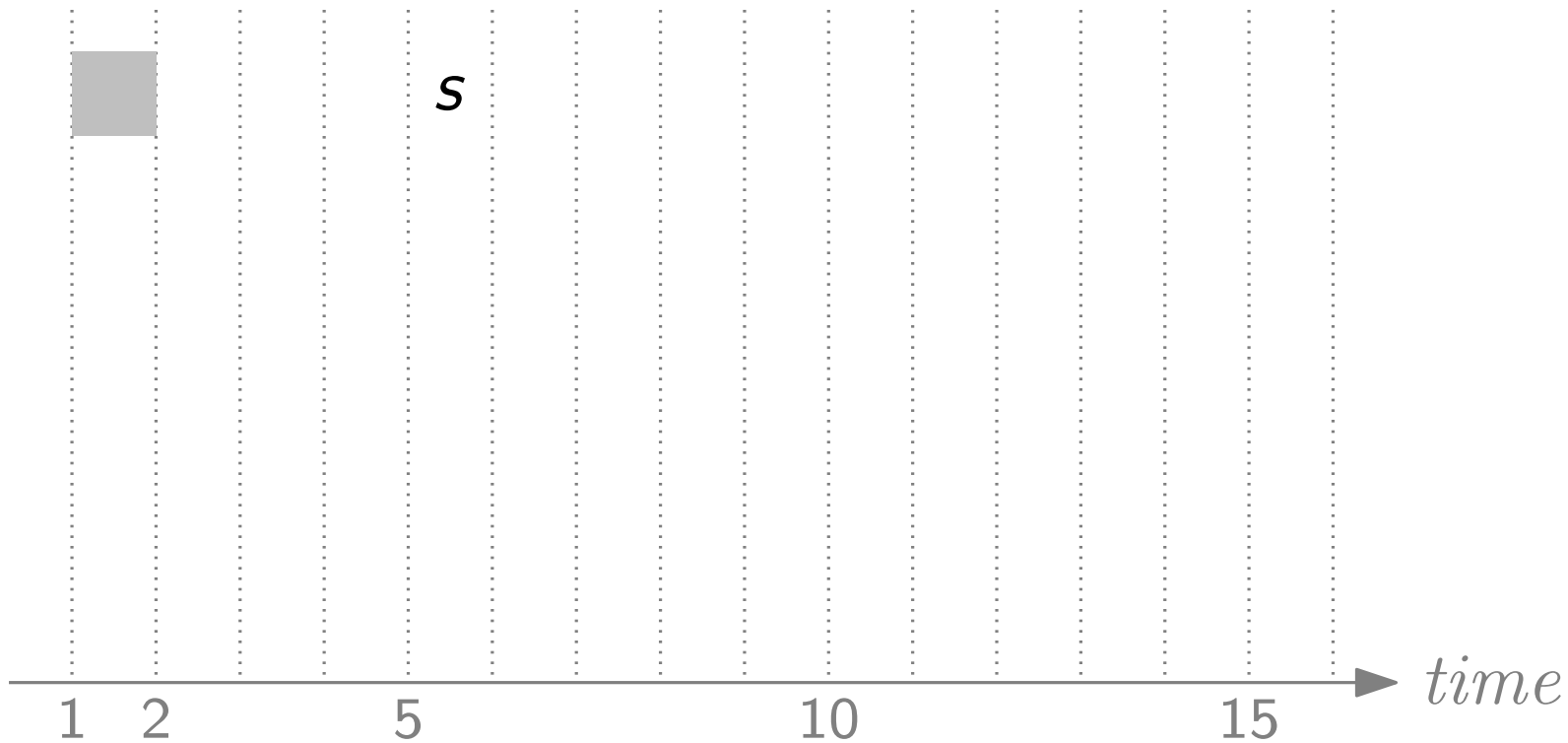
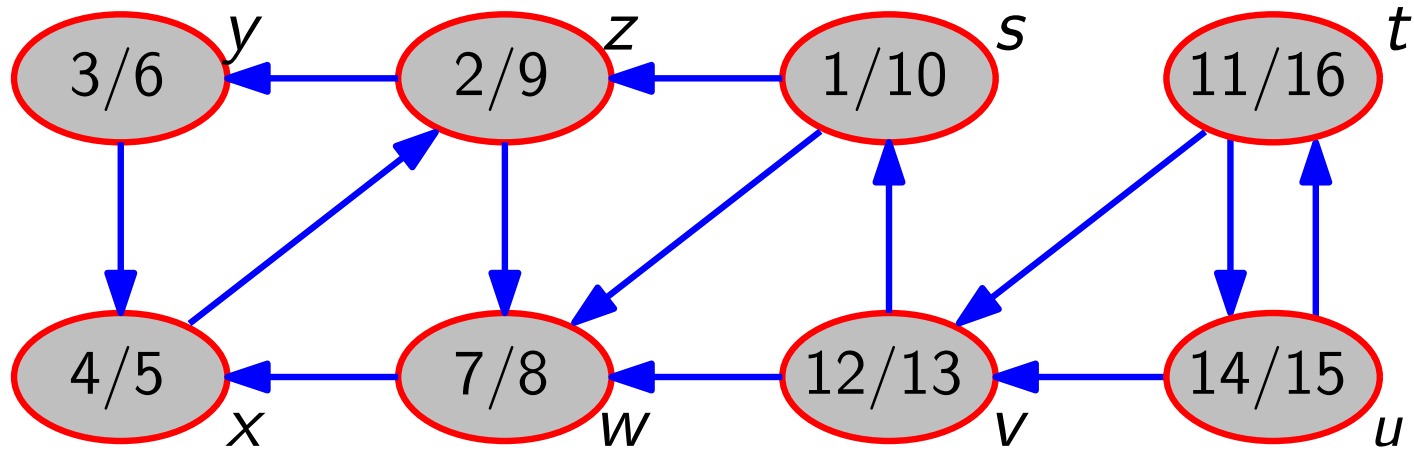


# Tiefensuche – Eigenschaften

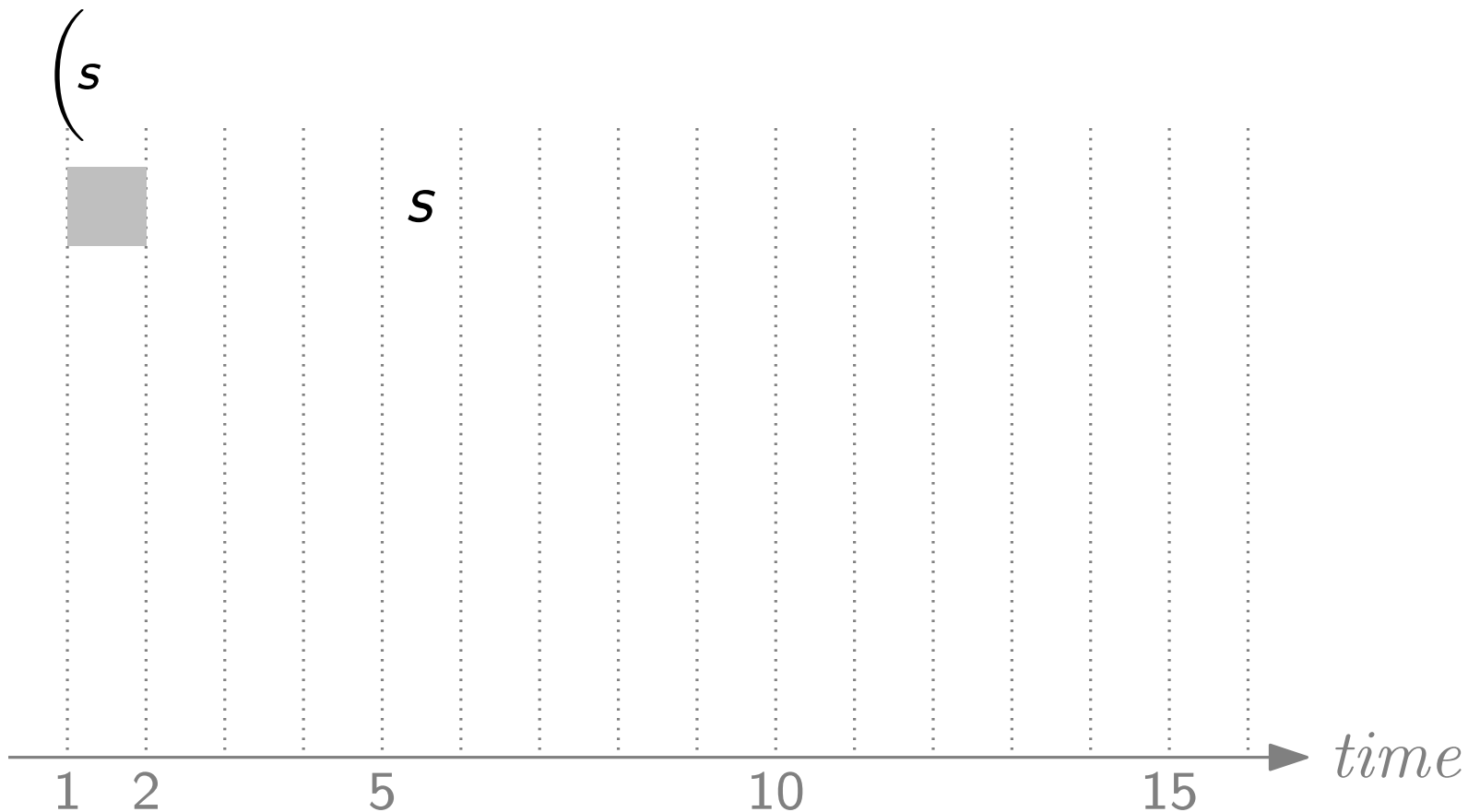
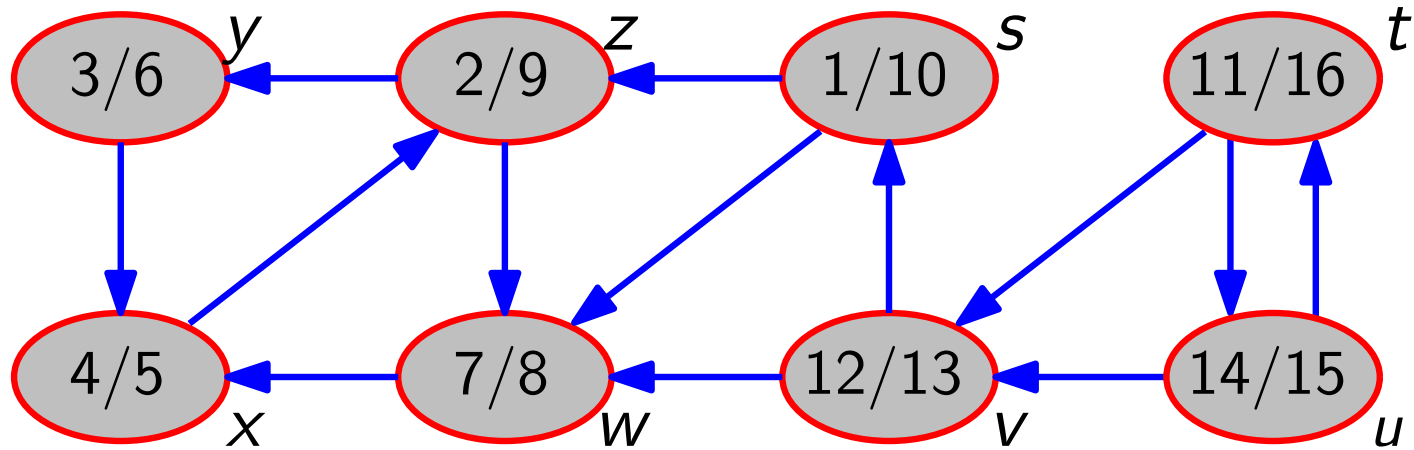




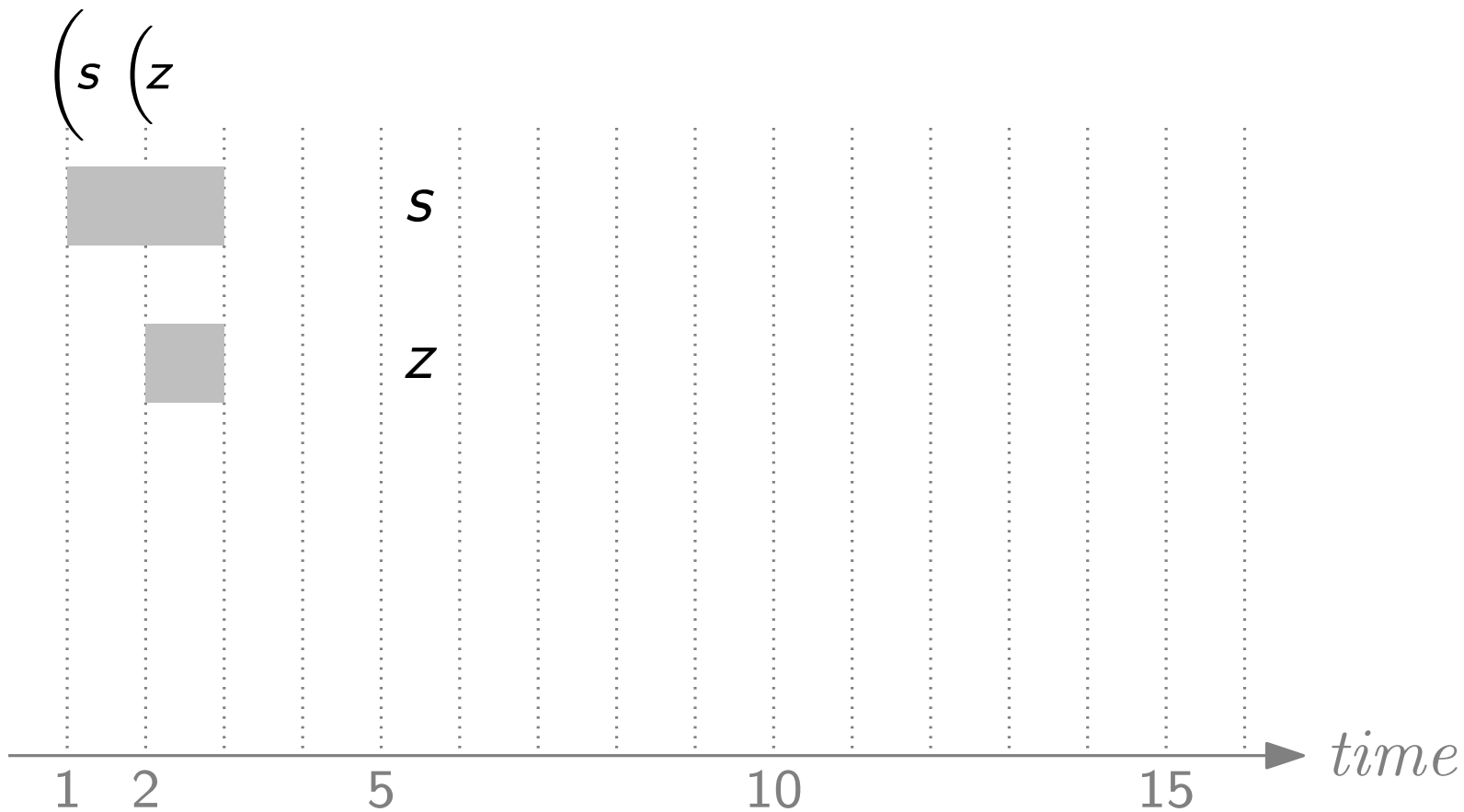
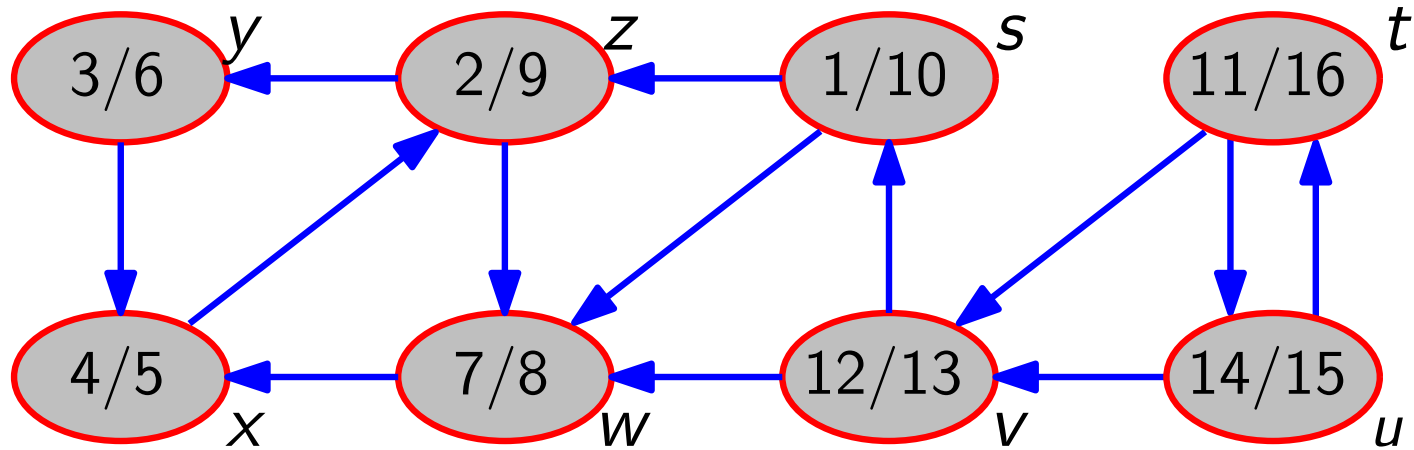
# Tiefensuche – Eigenschaften



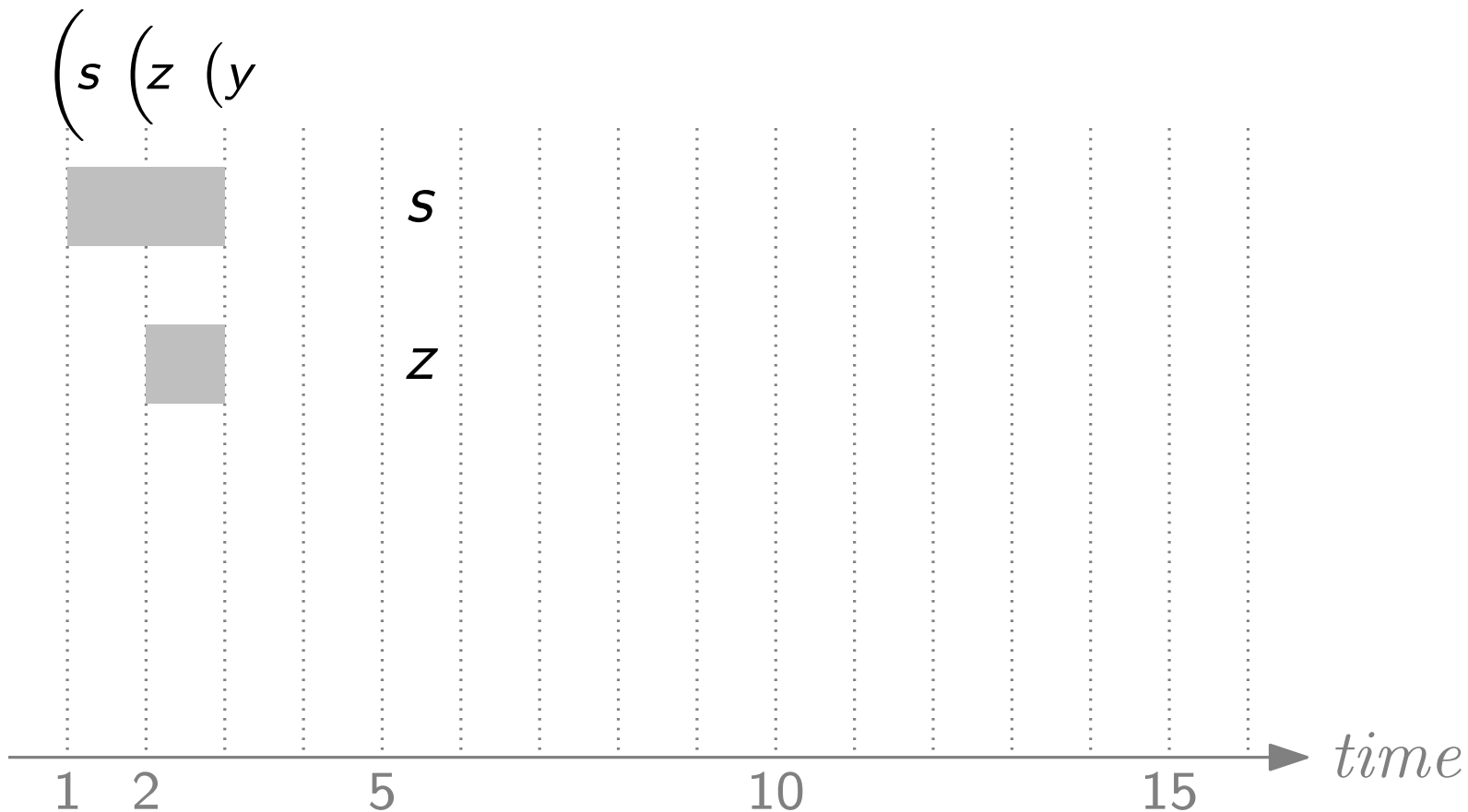
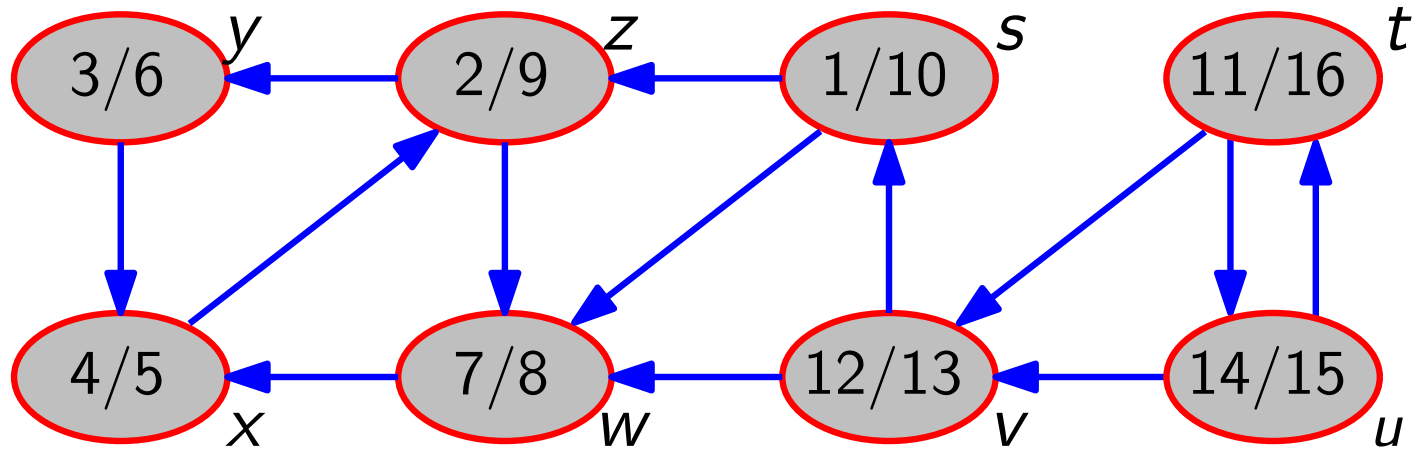
# Tiefensuche – Eigenschaften



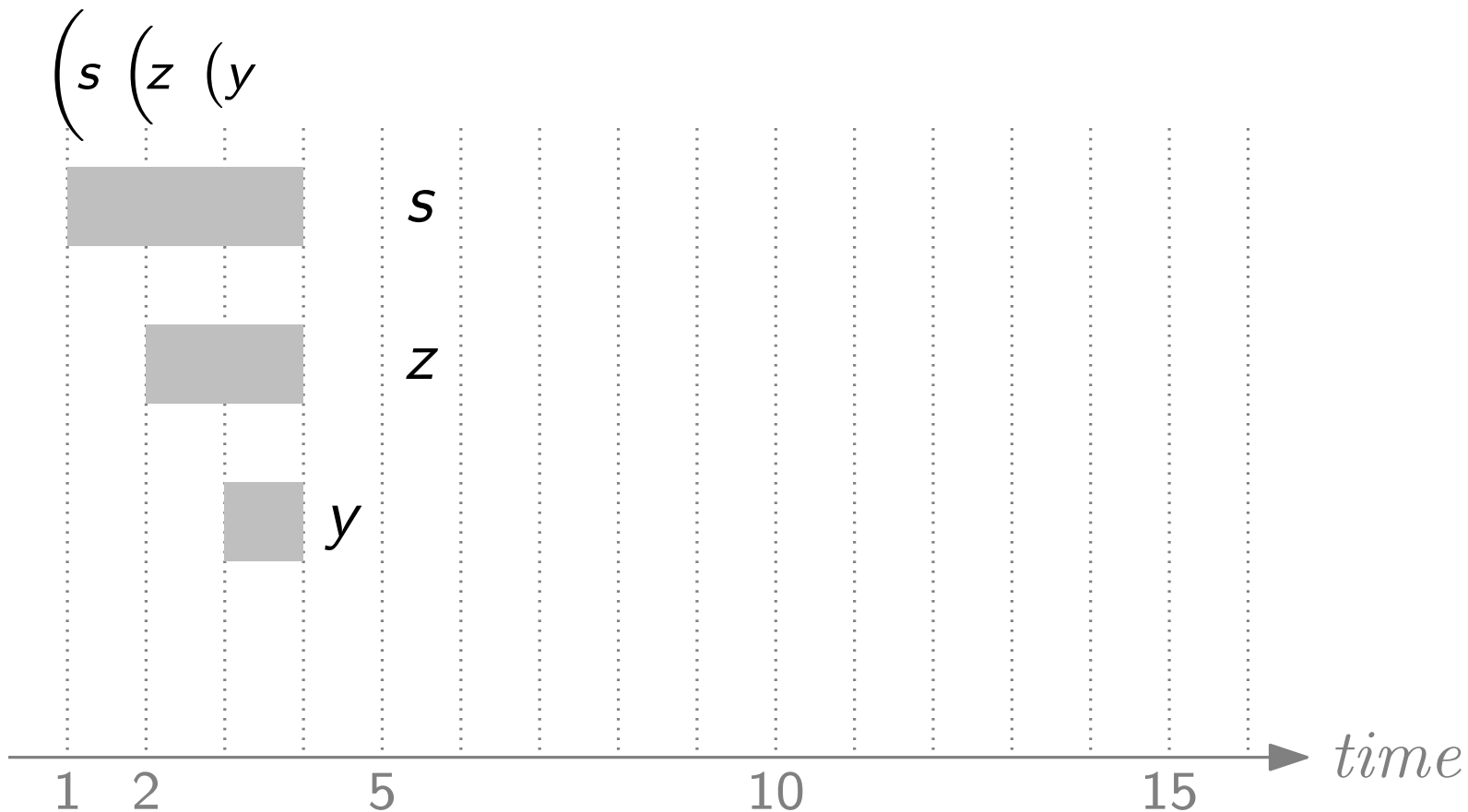
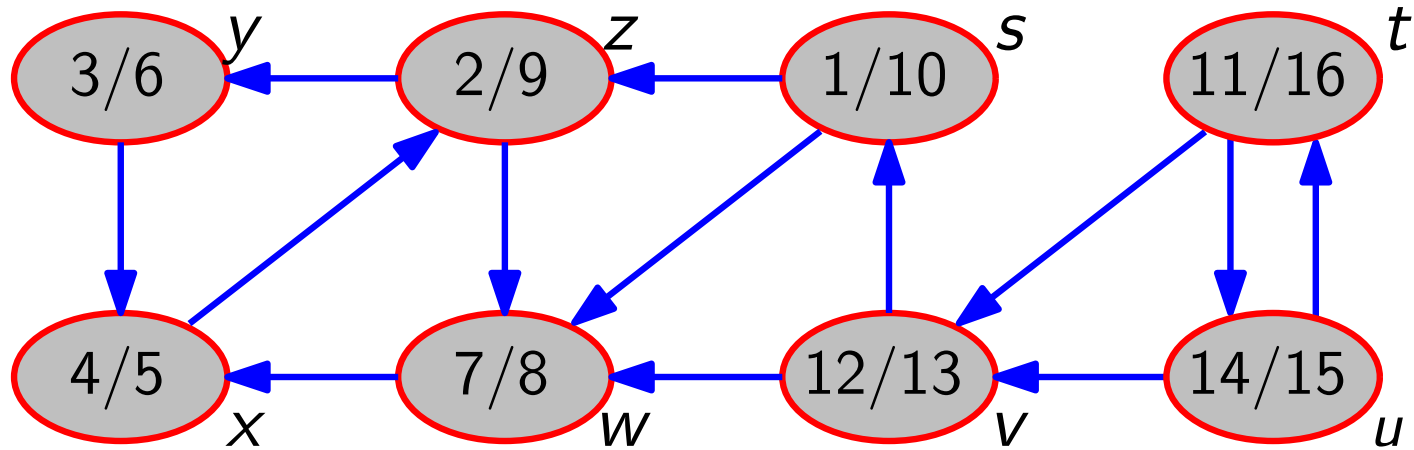
# Tiefensuche – Eigenschaften



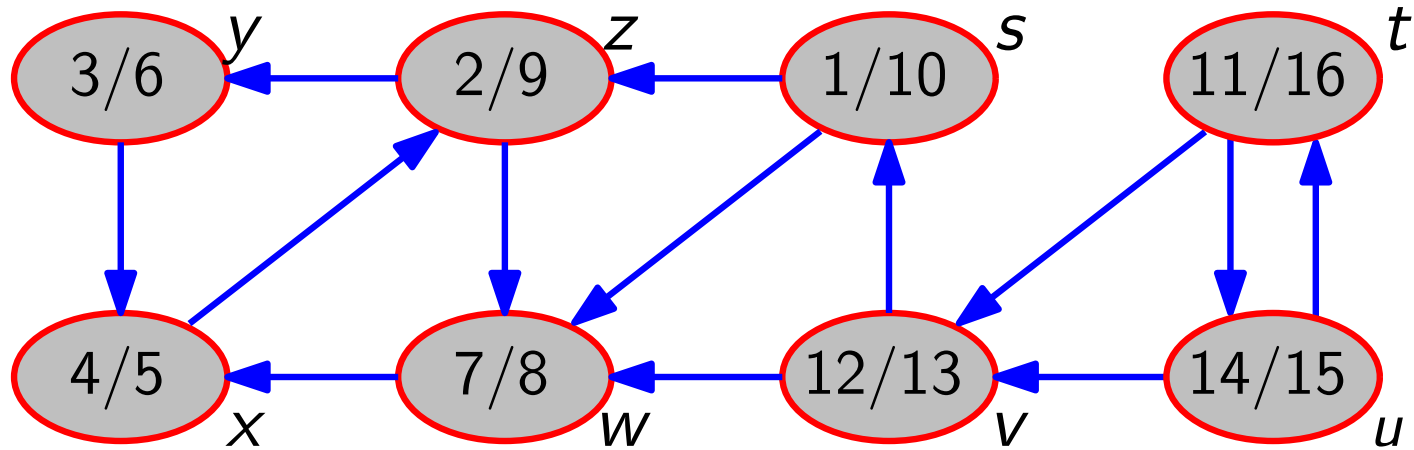
# Tiefensuche – Eigenschaften



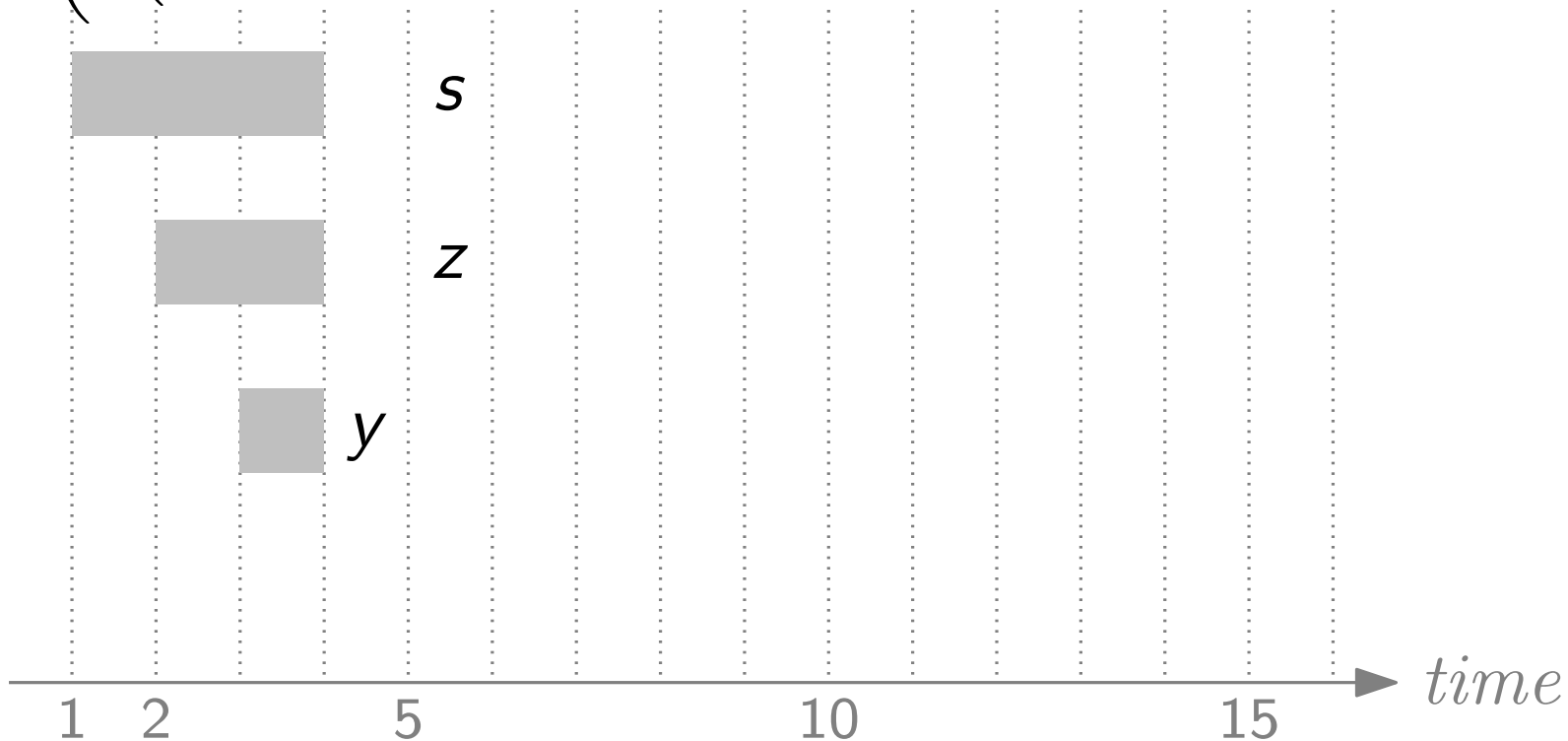
# Tiefensuche – Eigenschaften



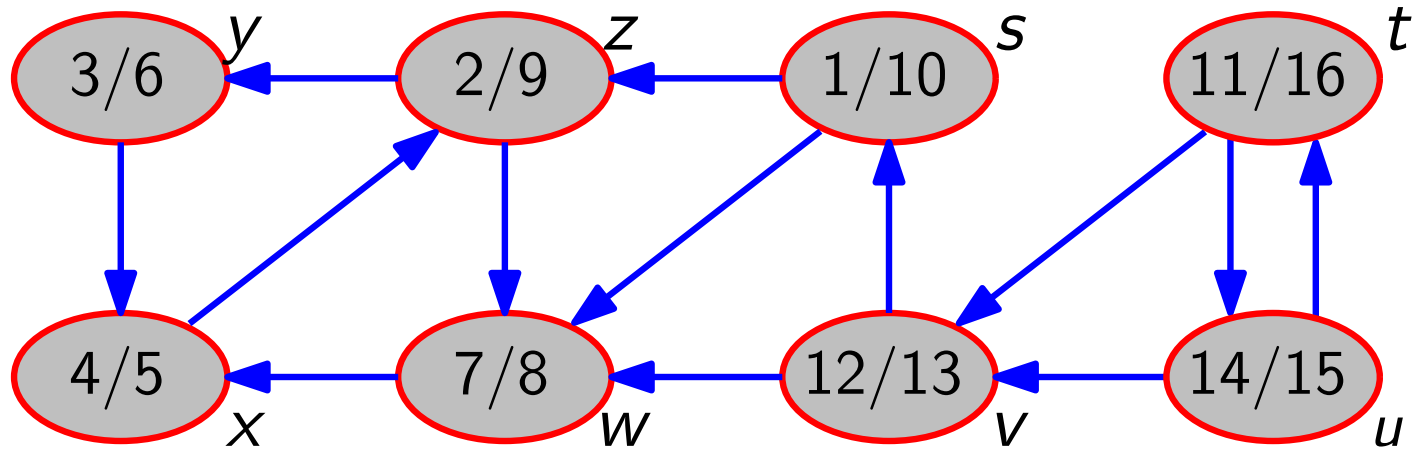
# Tiefensuche – Eigenschaften



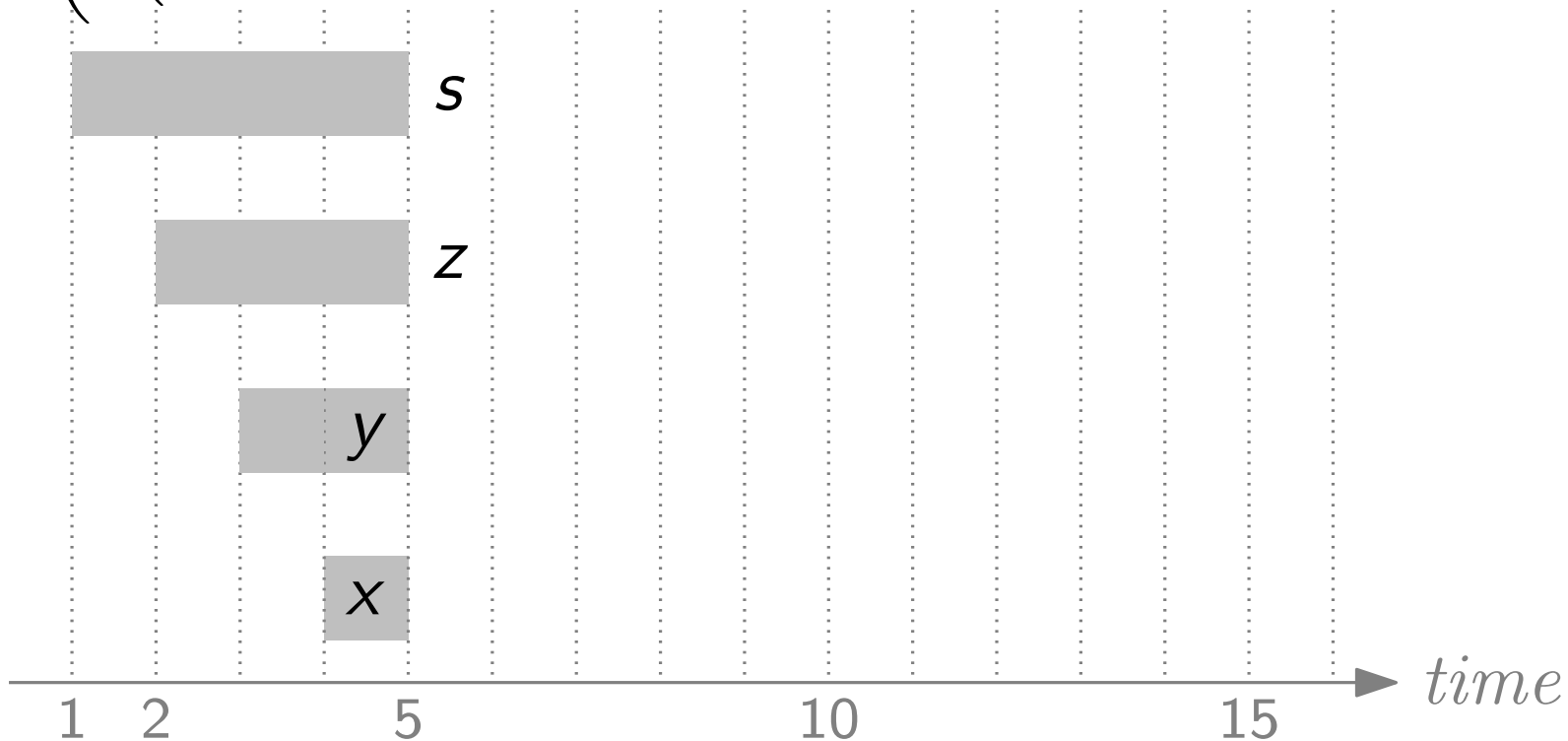
$(s (z (y (x$



# Tiefensuche – Eigenschaften



$(s (z (y (x$

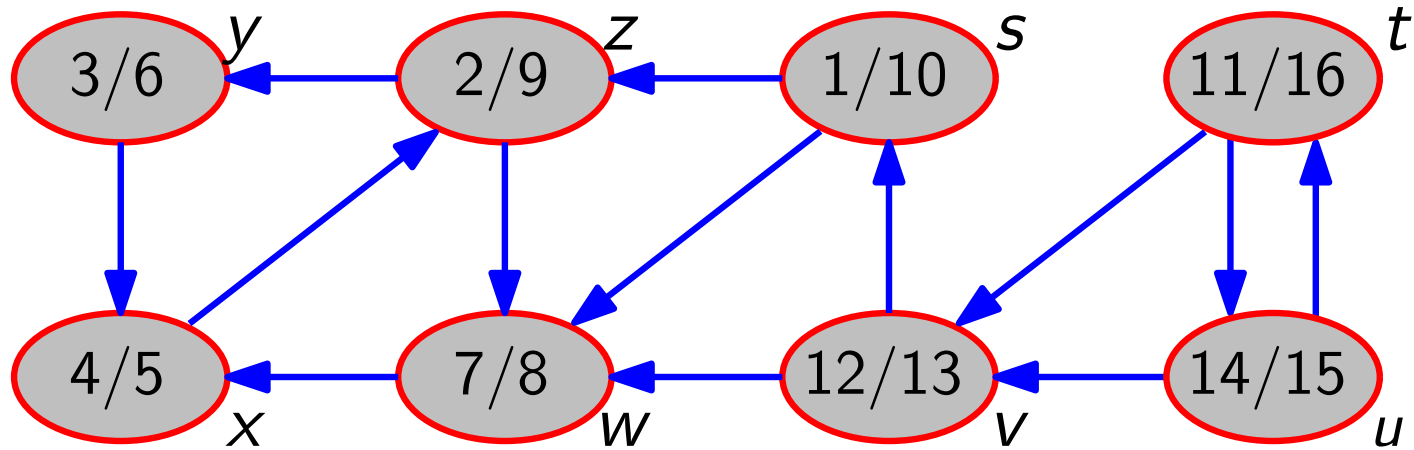




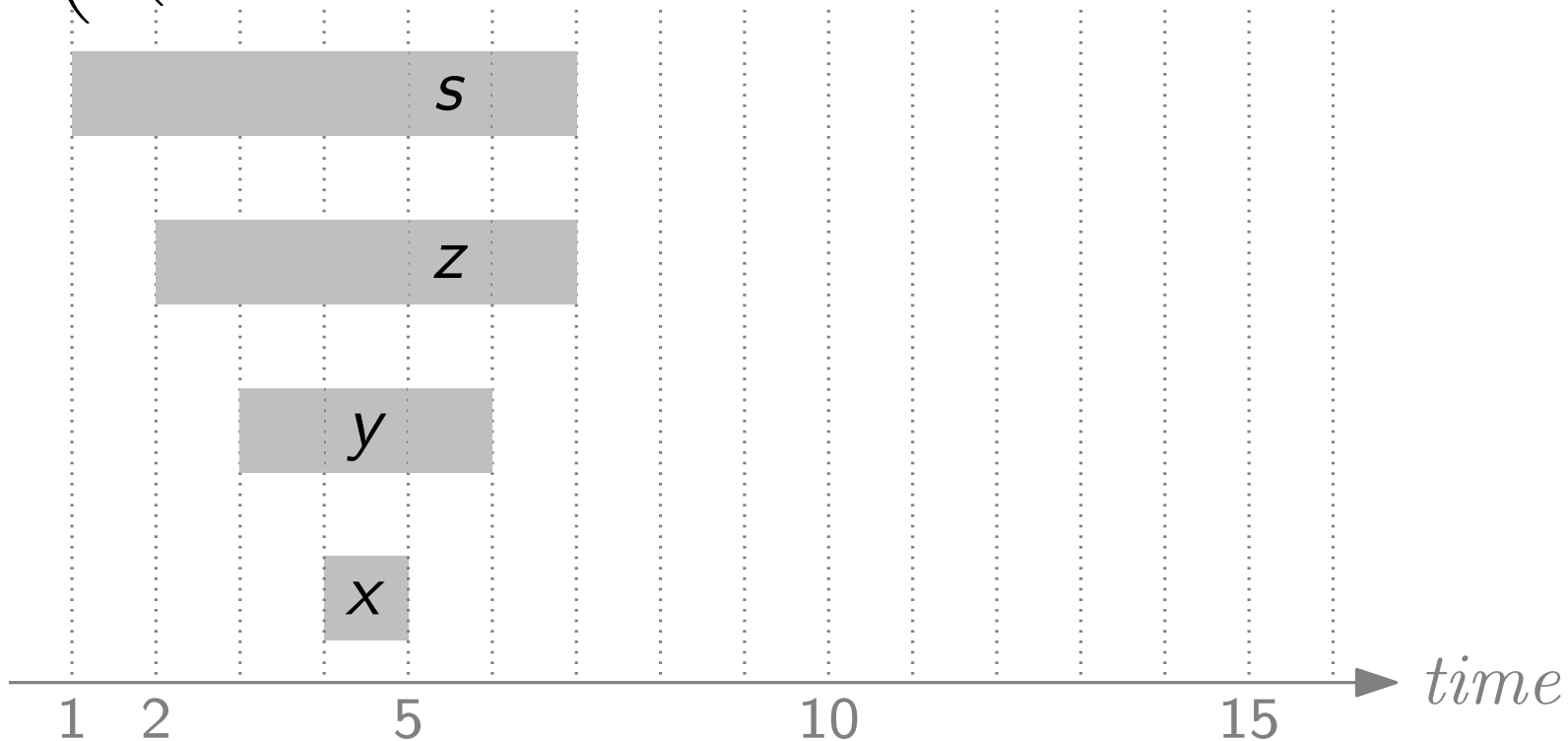




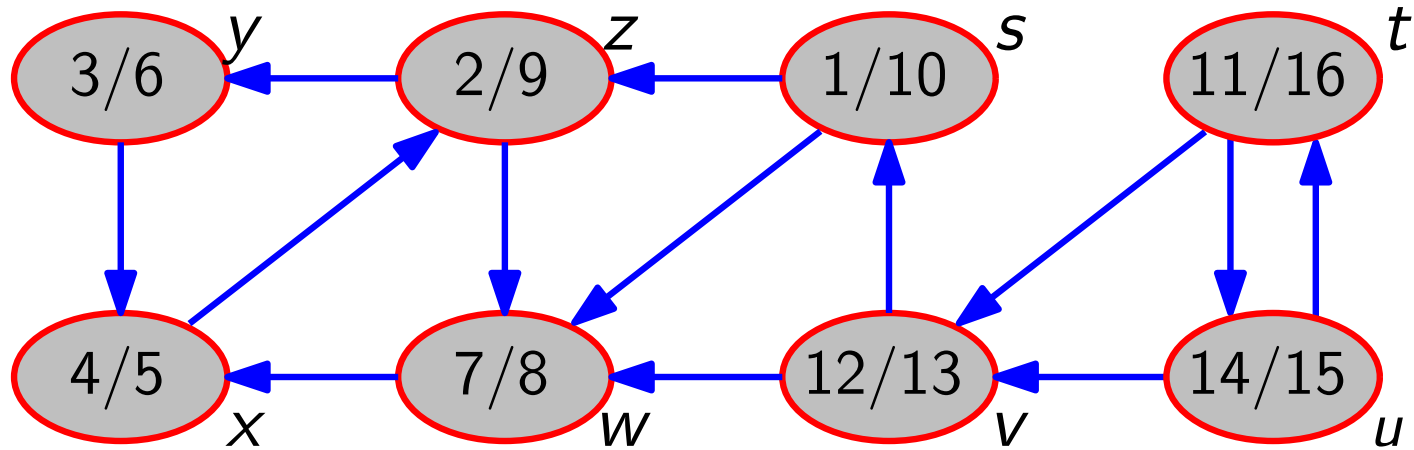
# Tiefensuche – Eigenschaften



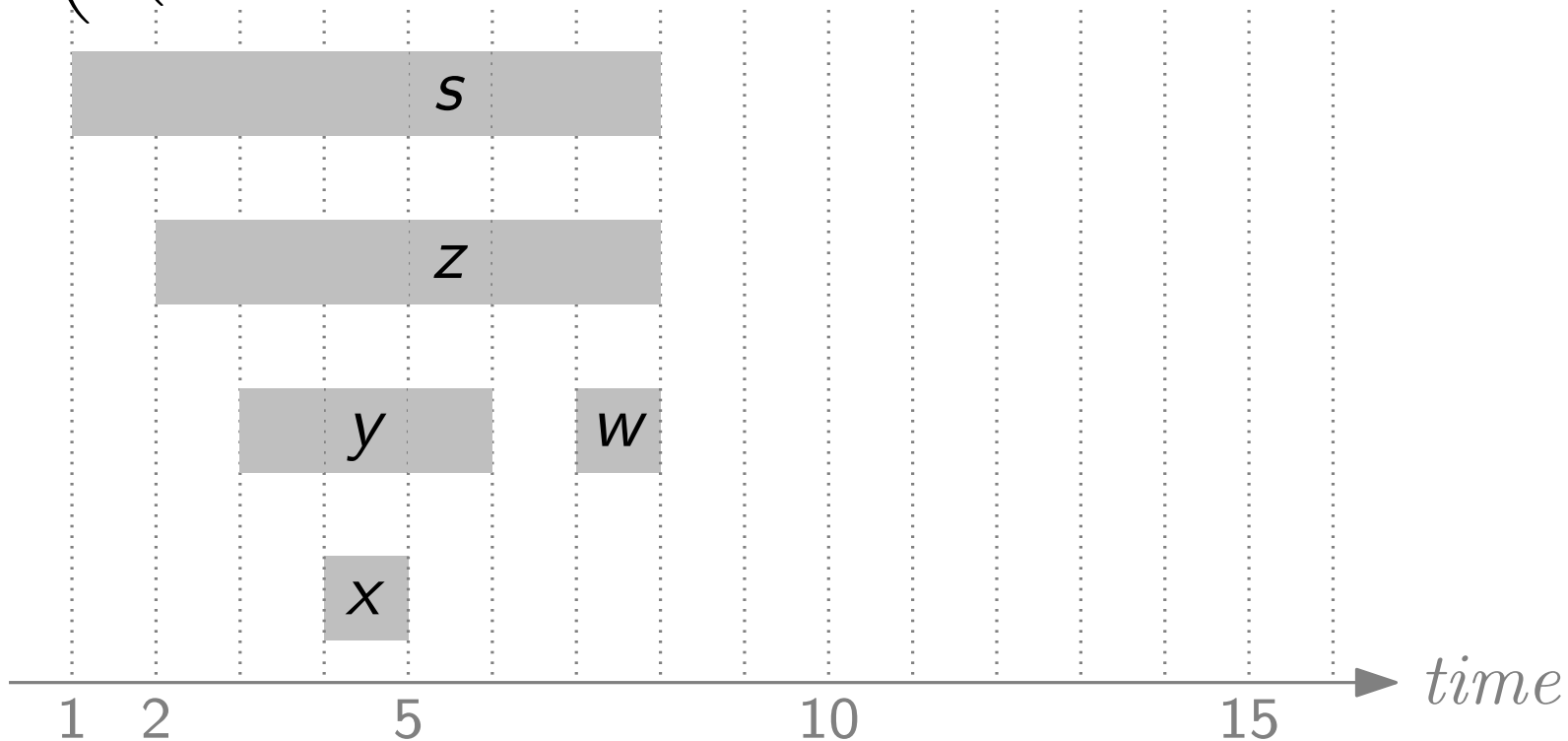
$(s (z (y (x x) y))$



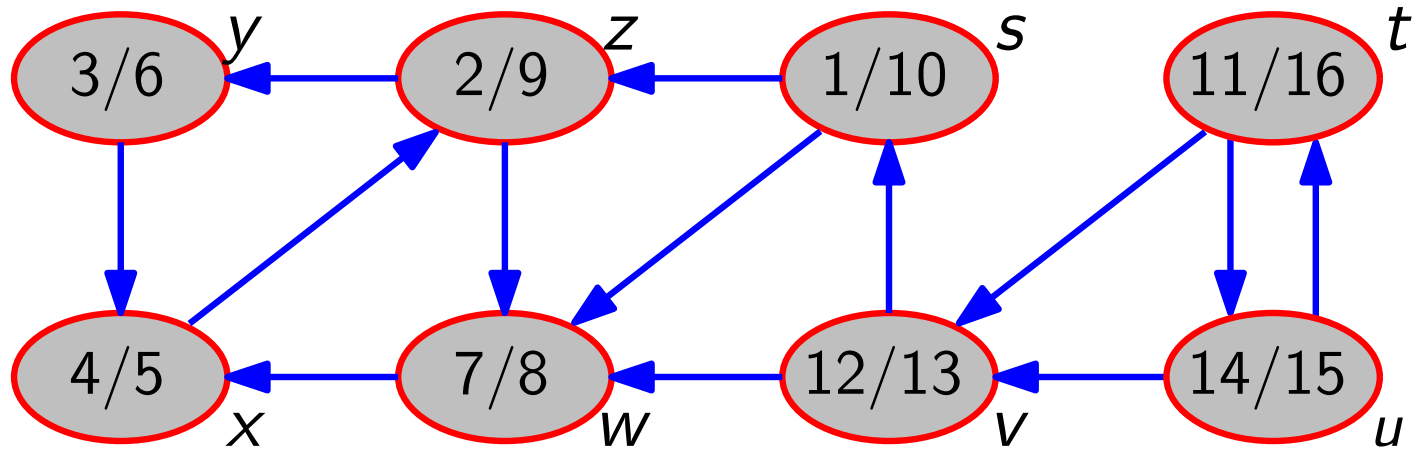
# Tiefensuche – Eigenschaften



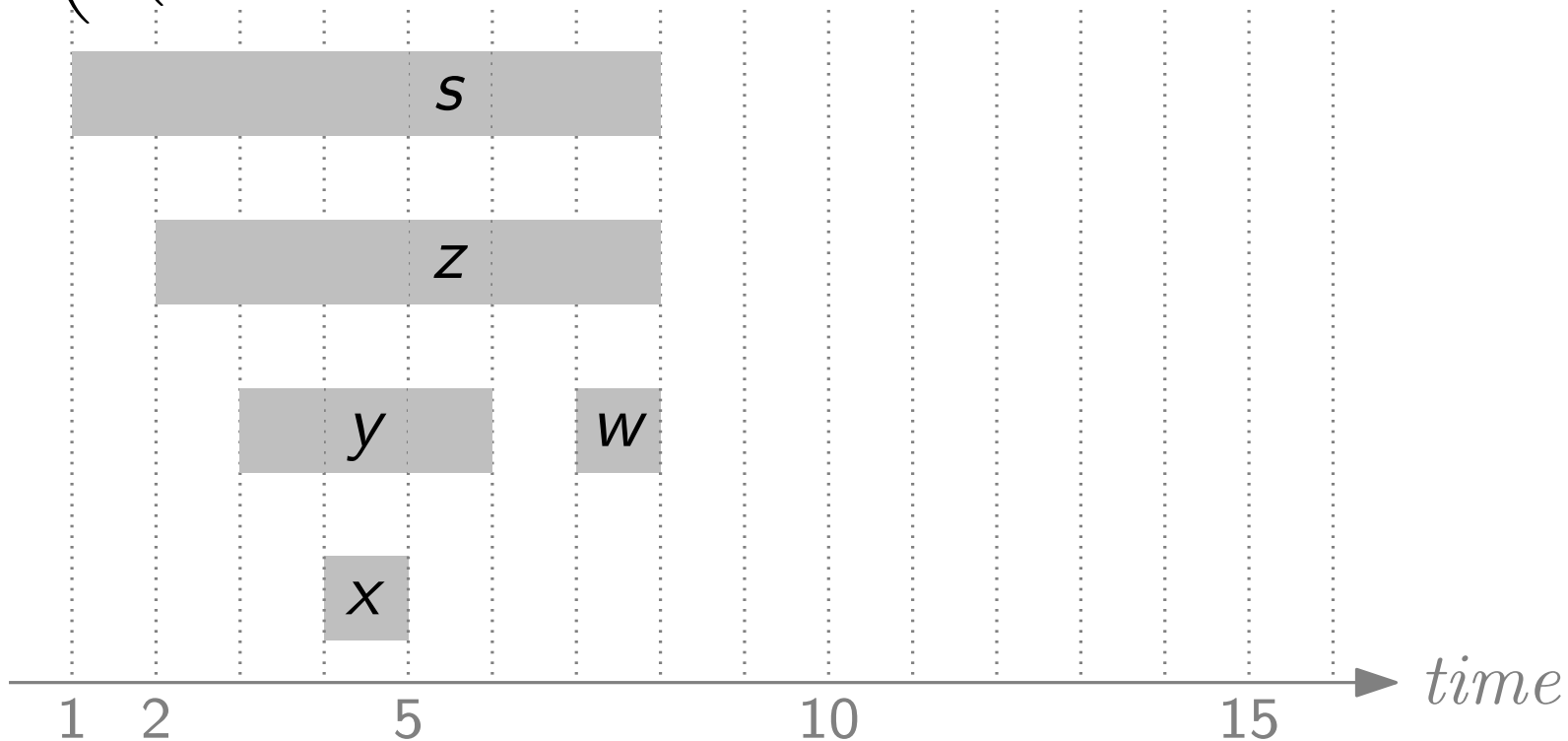
$(s (z (y (x x) y) (w$



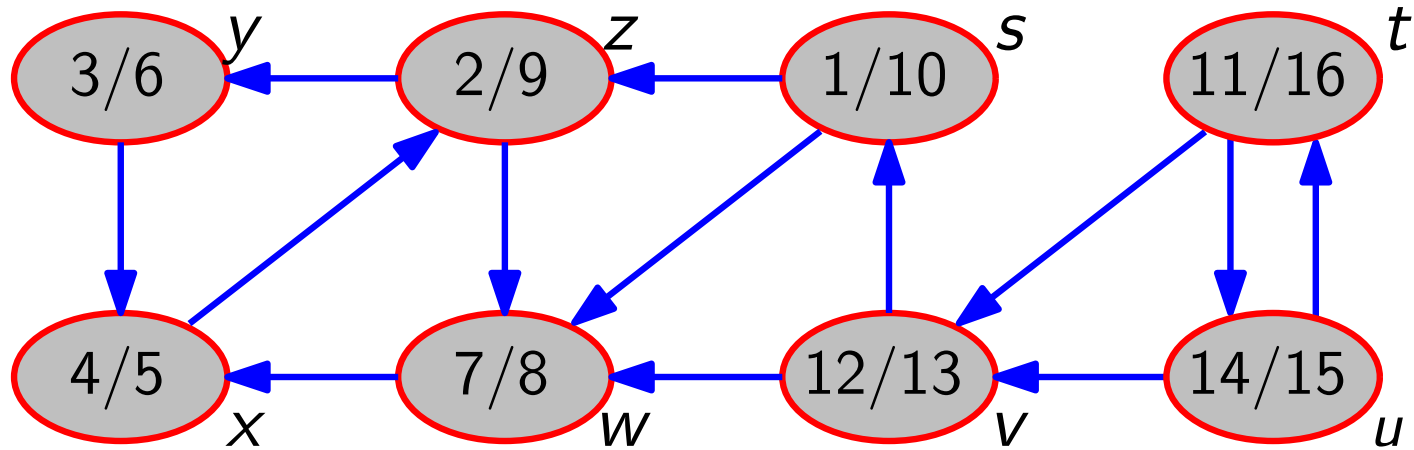
# Tiefensuche – Eigenschaften



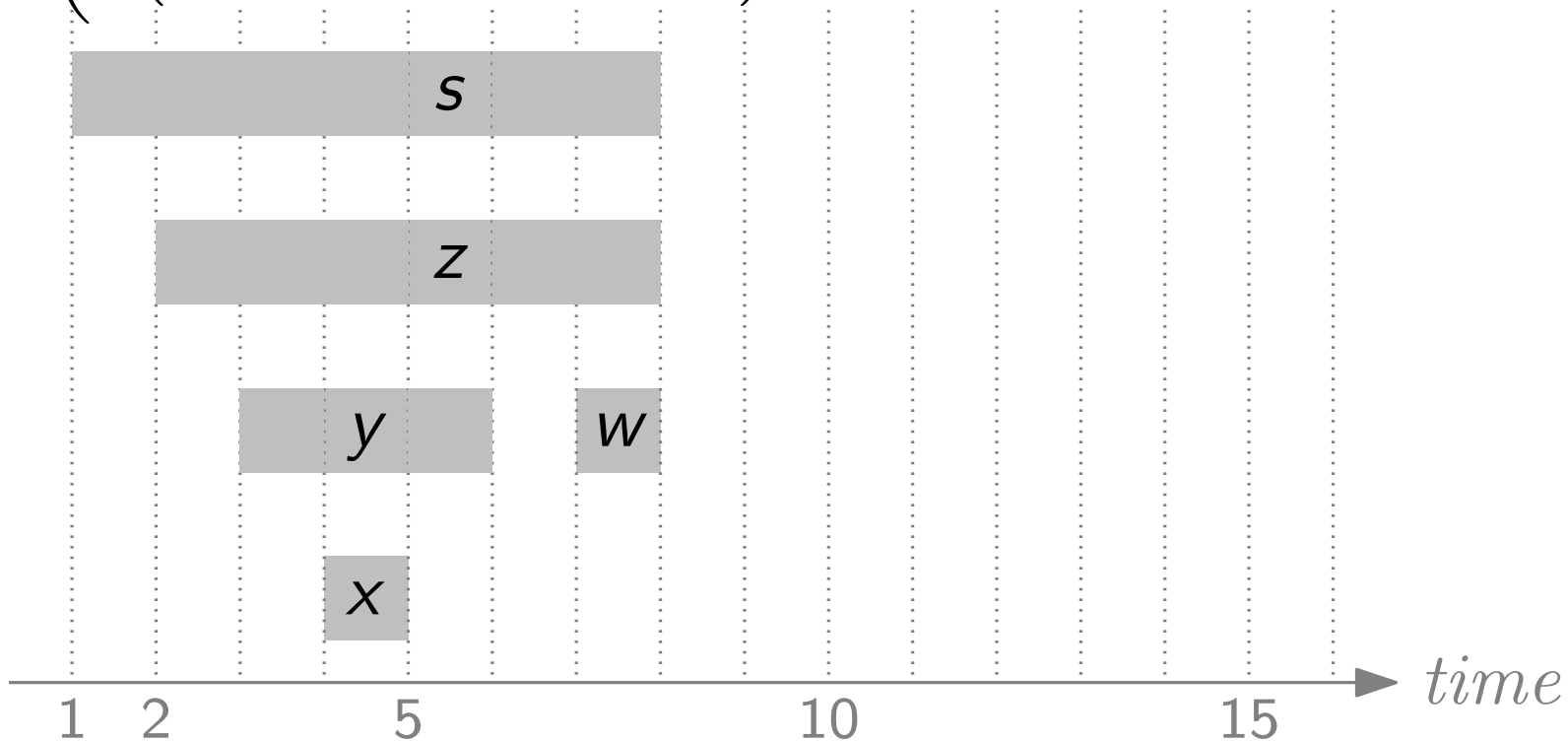
$(s (z (y (x x) y) (w w)))$



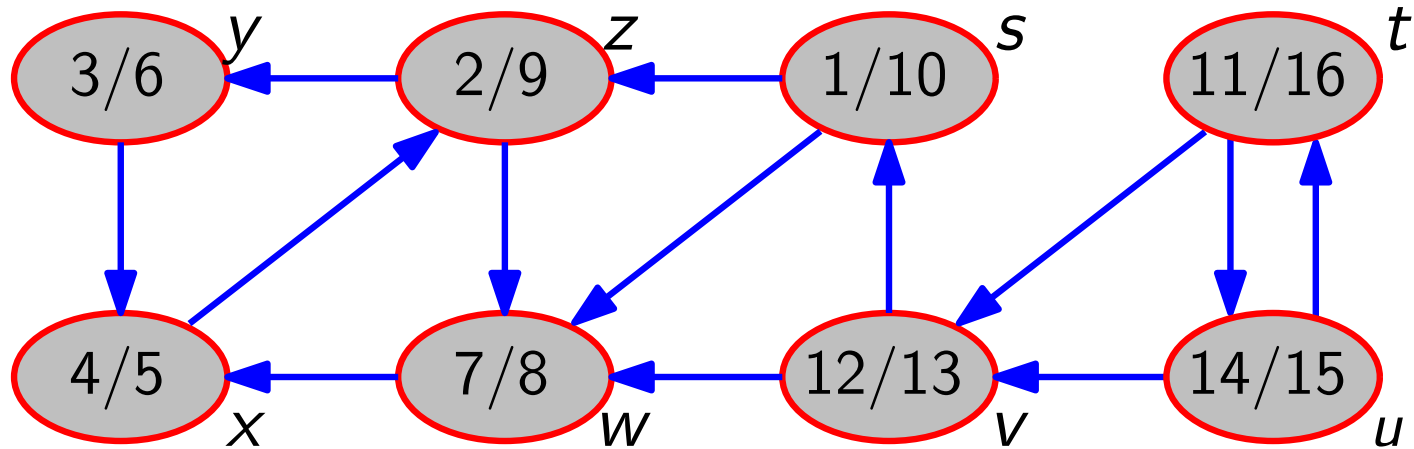
# Tiefensuche – Eigenschaften



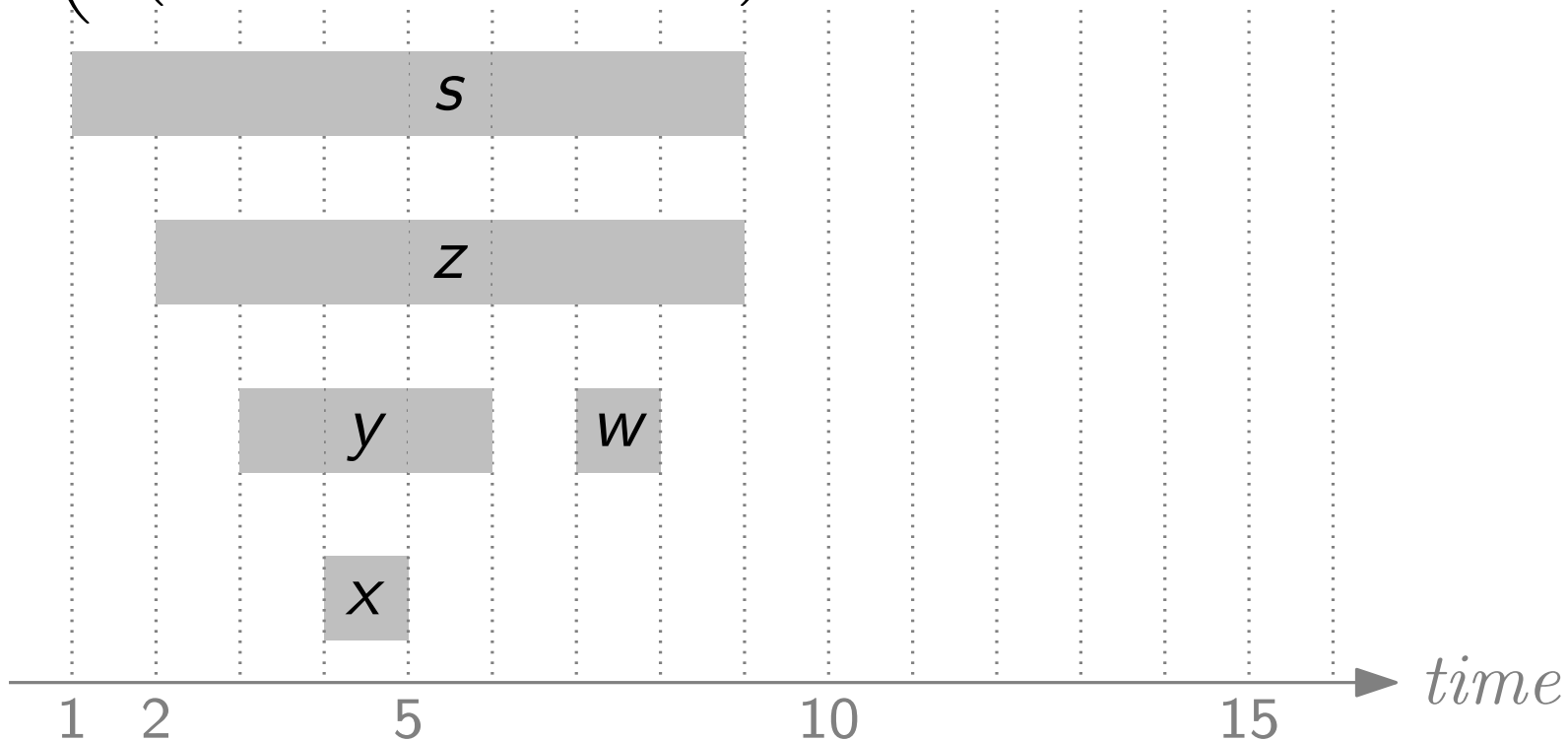
$(s (z (y (x x) y) (w w) z))$



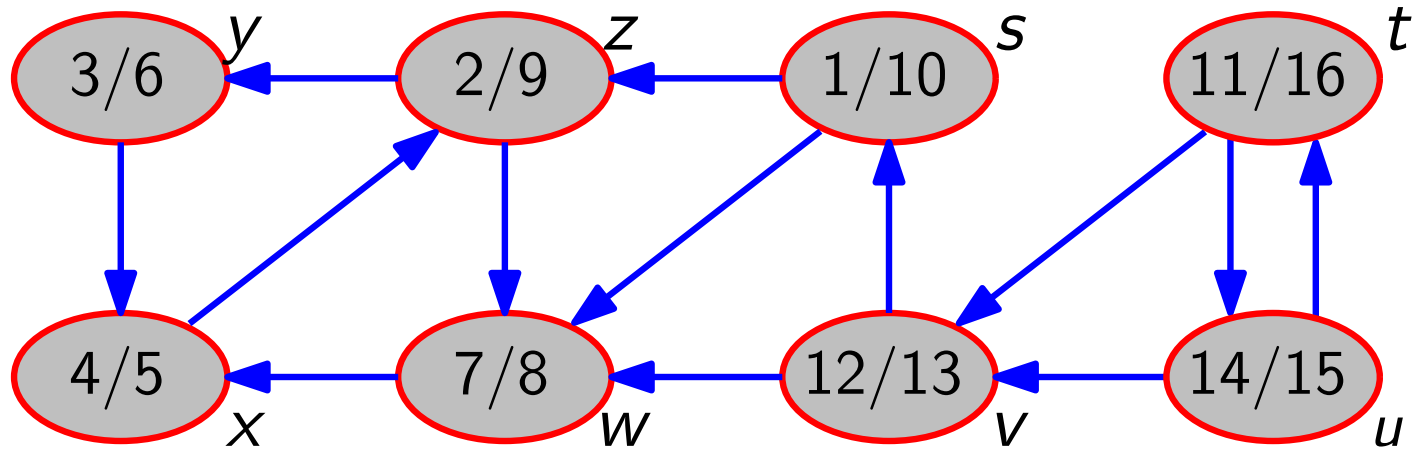
# Tiefensuche – Eigenschaften



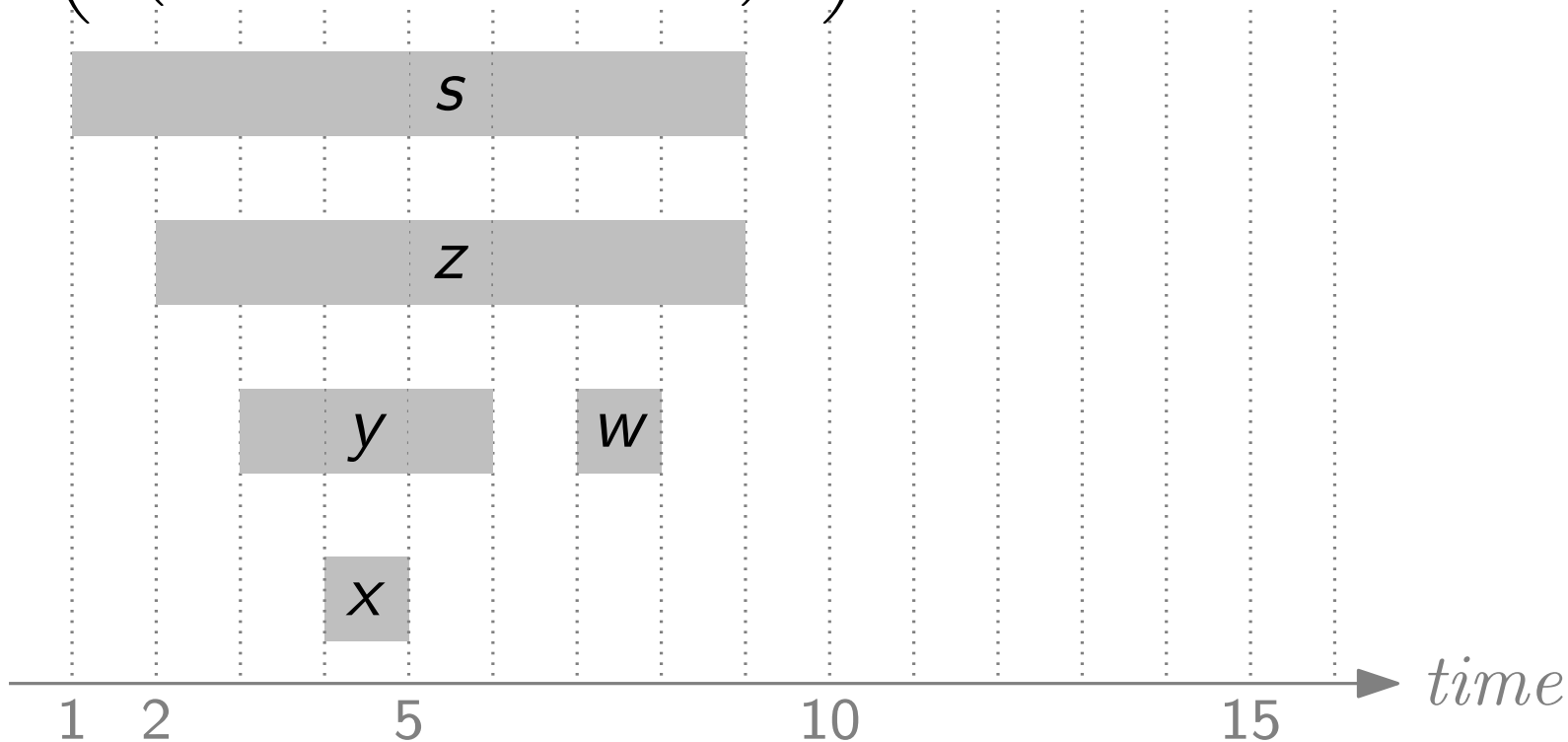
$(s (z (y (x x) y) (w w) z))$



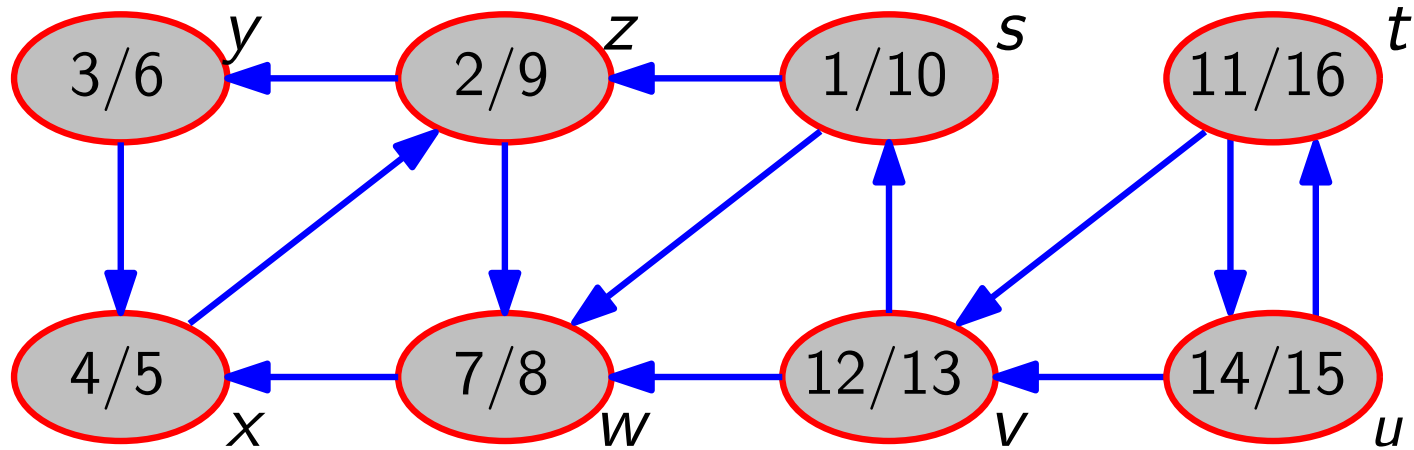
# Tiefensuche – Eigenschaften



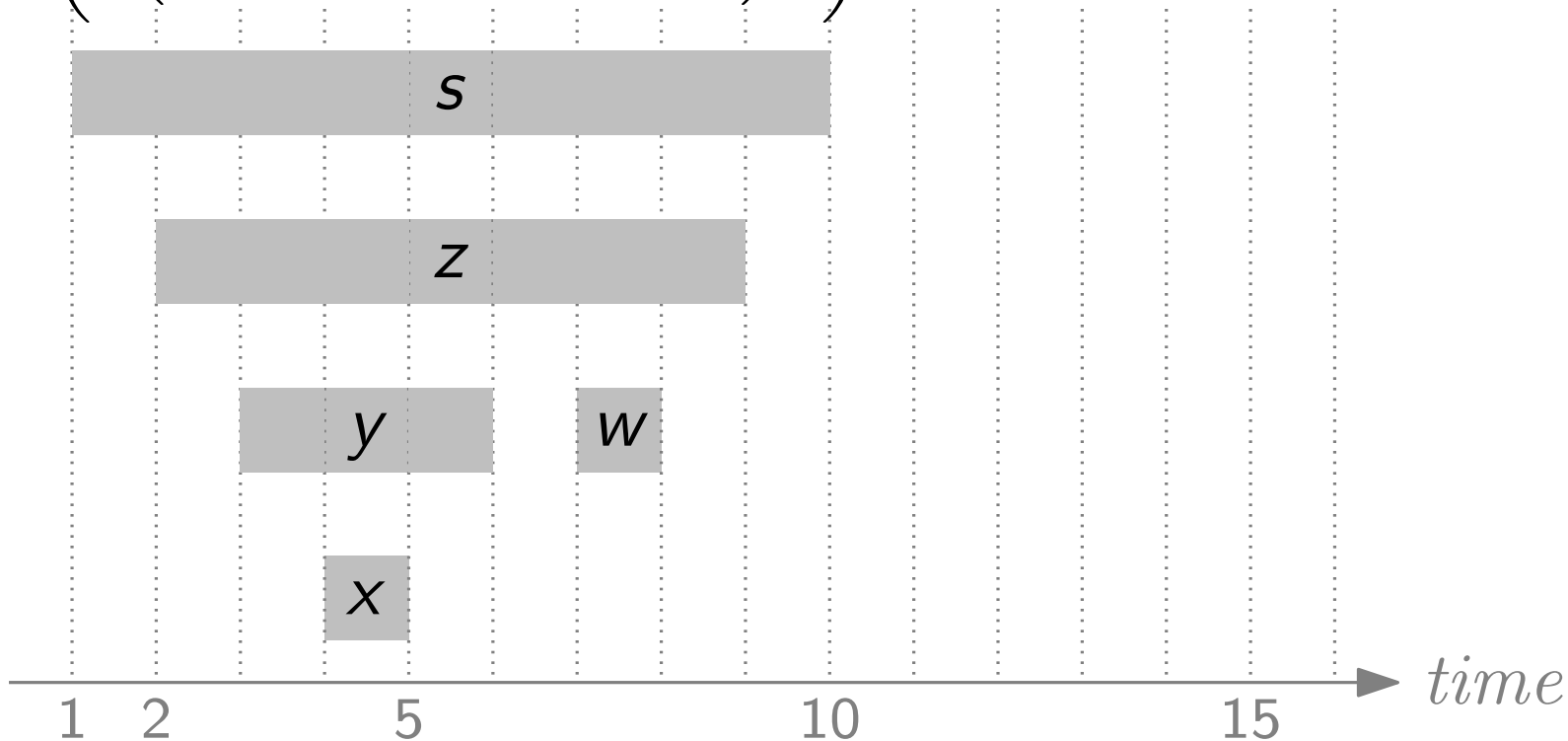
$(s (z (y (x x) y) (w w) z) s)$



# Tiefensuche – Eigenschaften

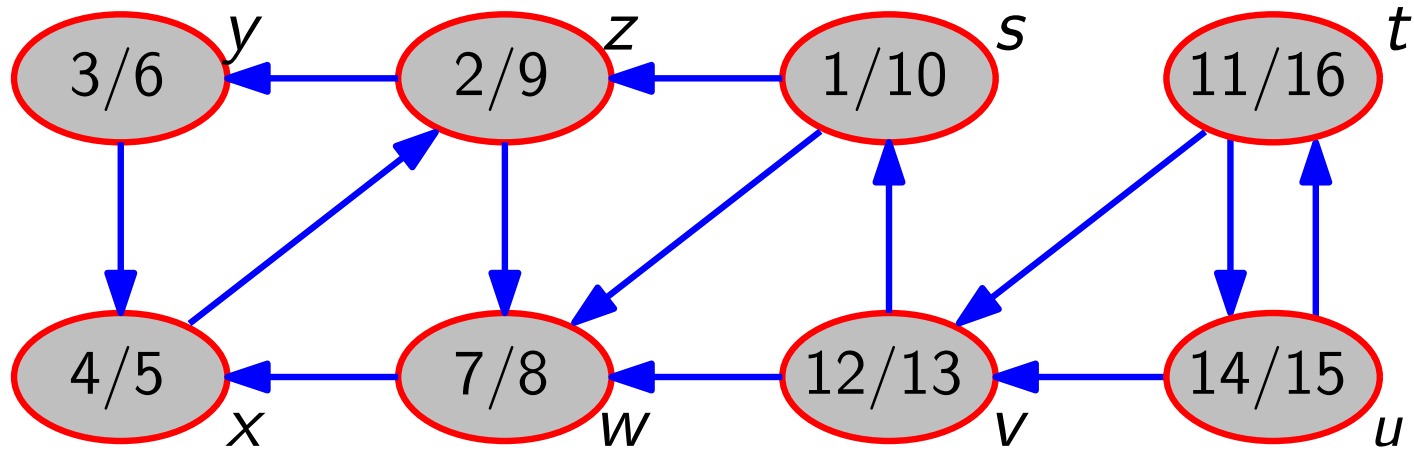


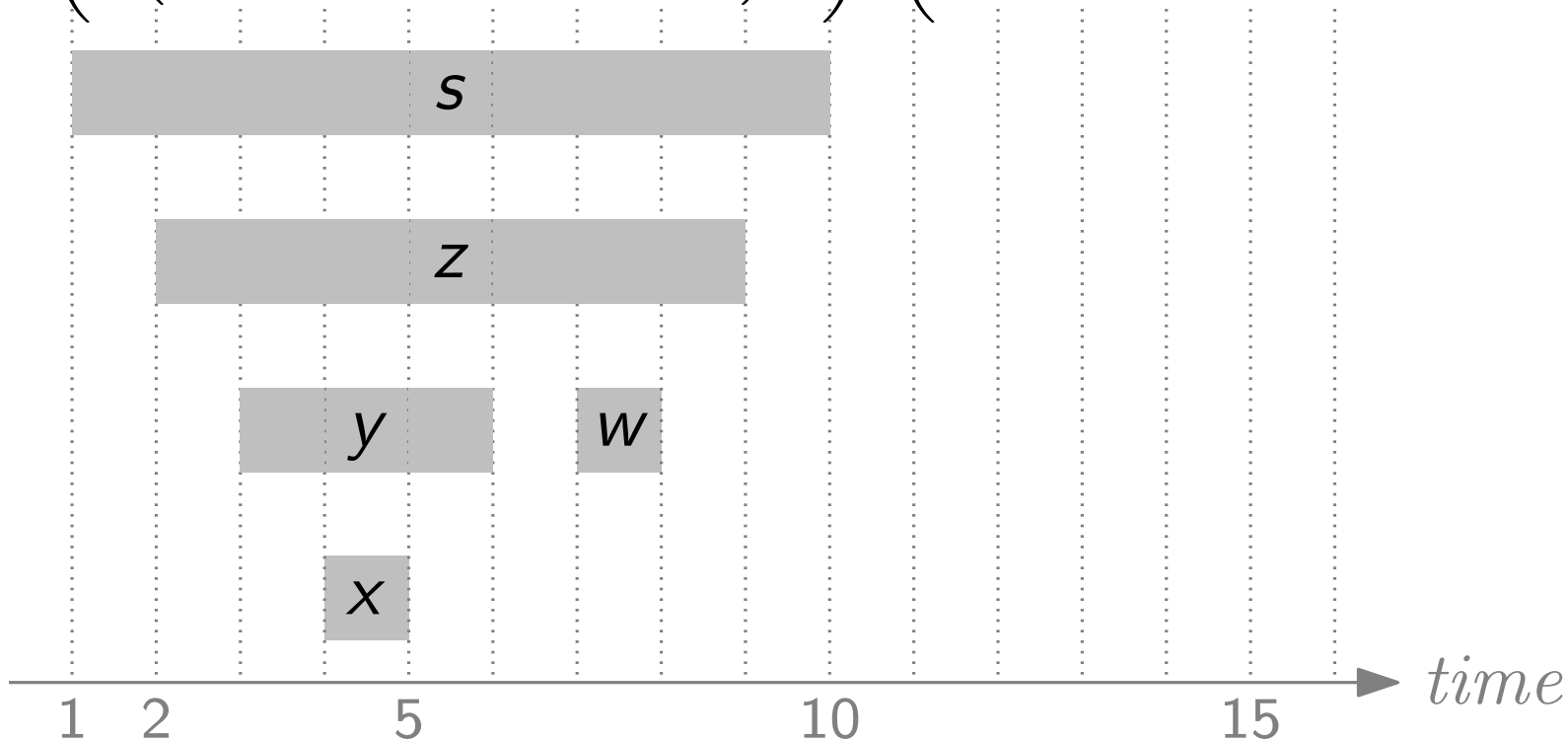
$(s (z (y (x x) y) (w w) z) s)$



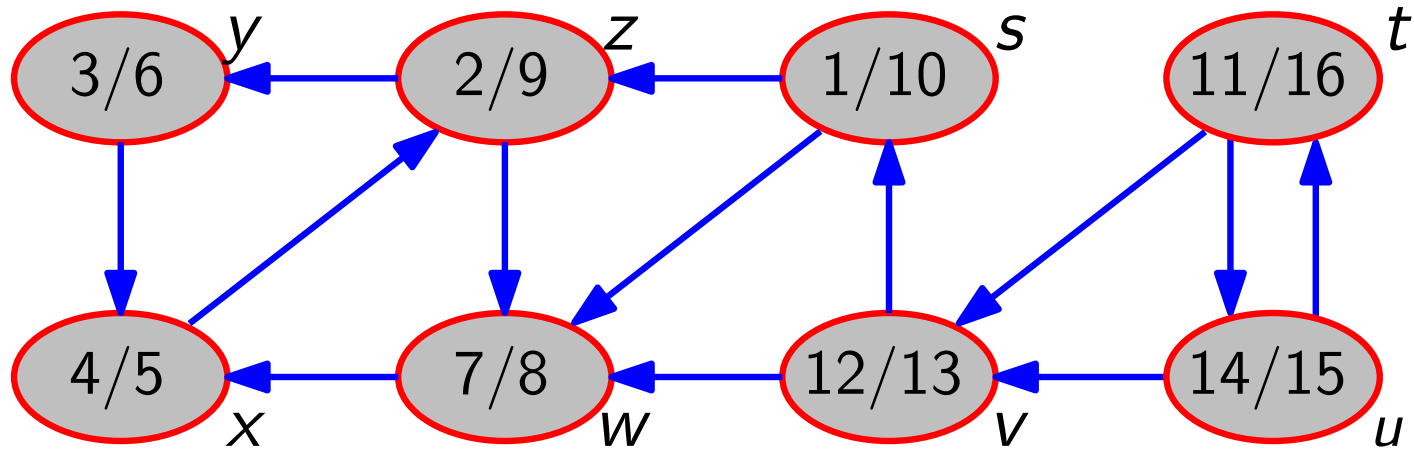


# Tiefensuche – Eigenschaften

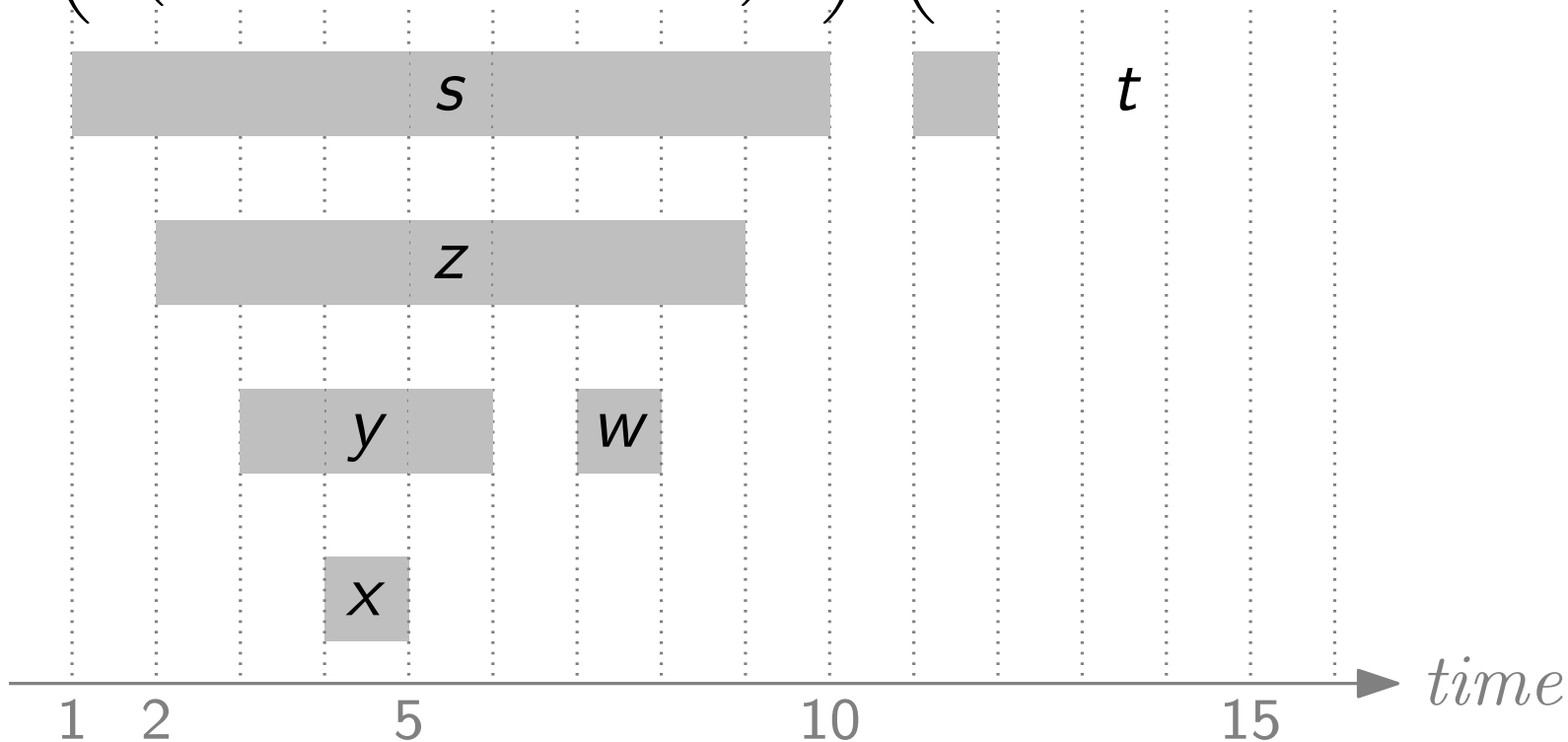


$$\left( s \left( z \left( y \left( x \ x \right) y \right) \left( w \ w \right) z \right) s \right) \left( t \right)$$


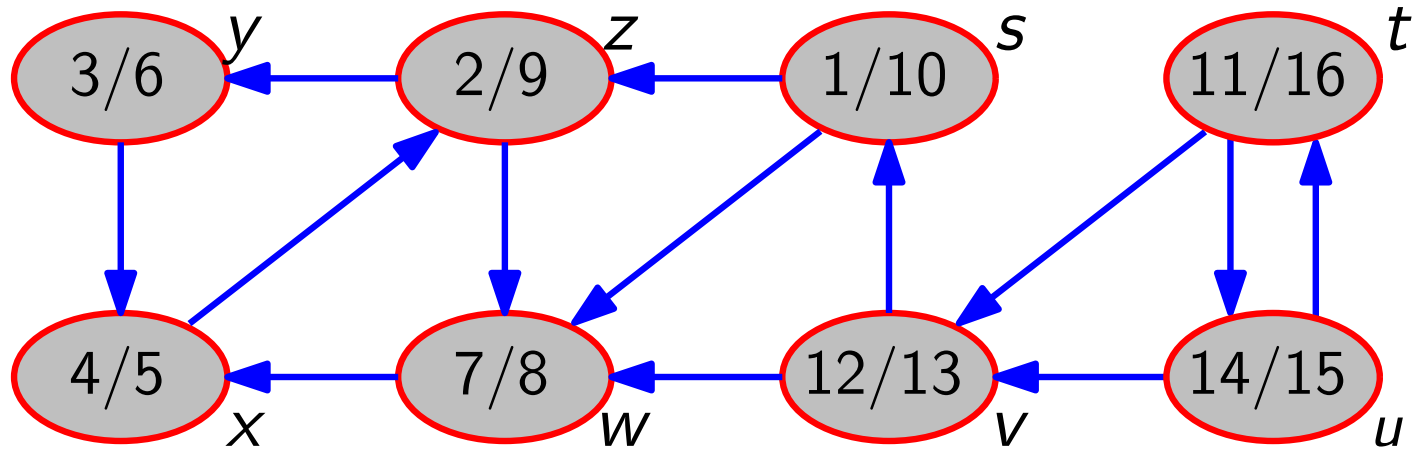
# Tiefensuche – Eigenschaften



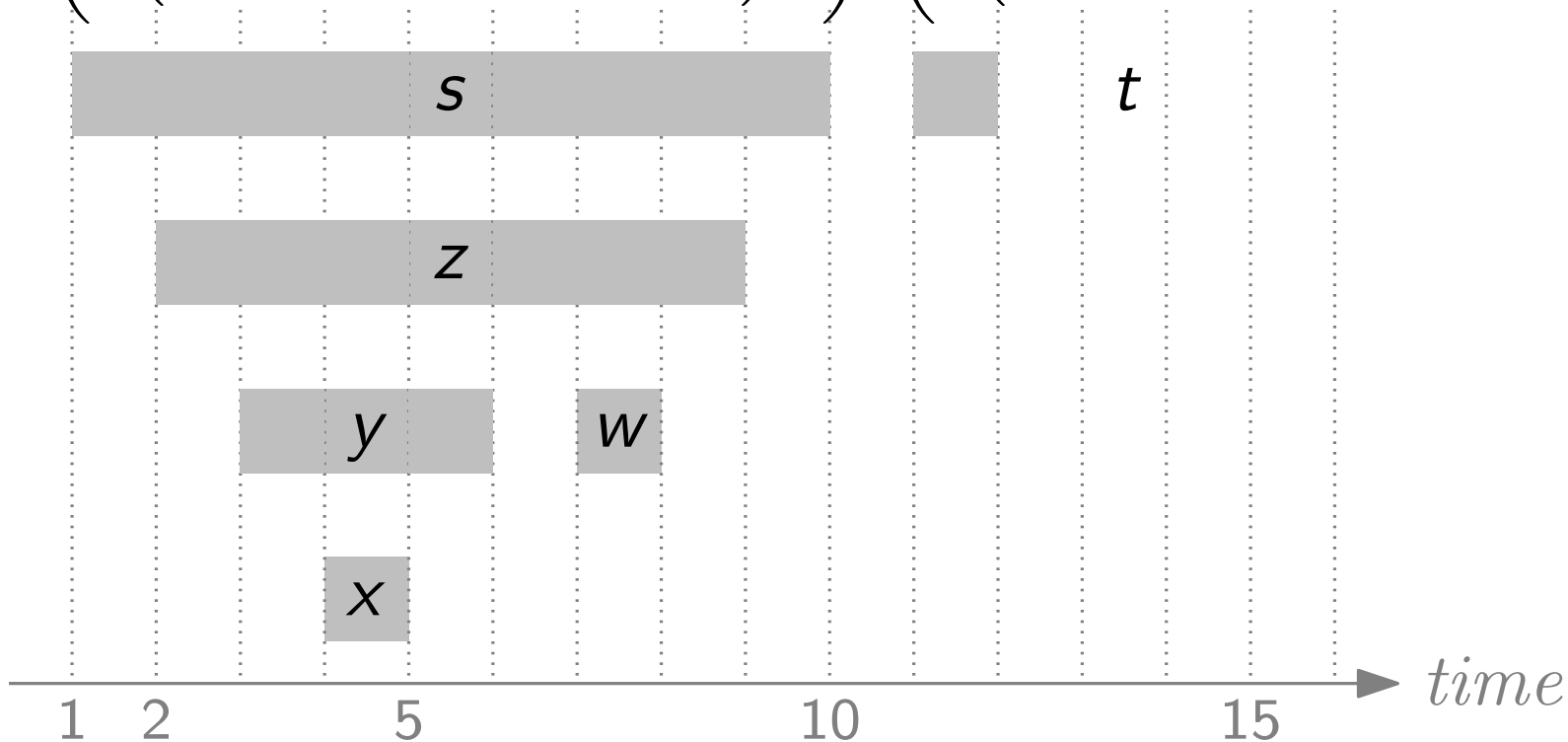
$(s (z (y (x x) y) (w w) z) s) (t$



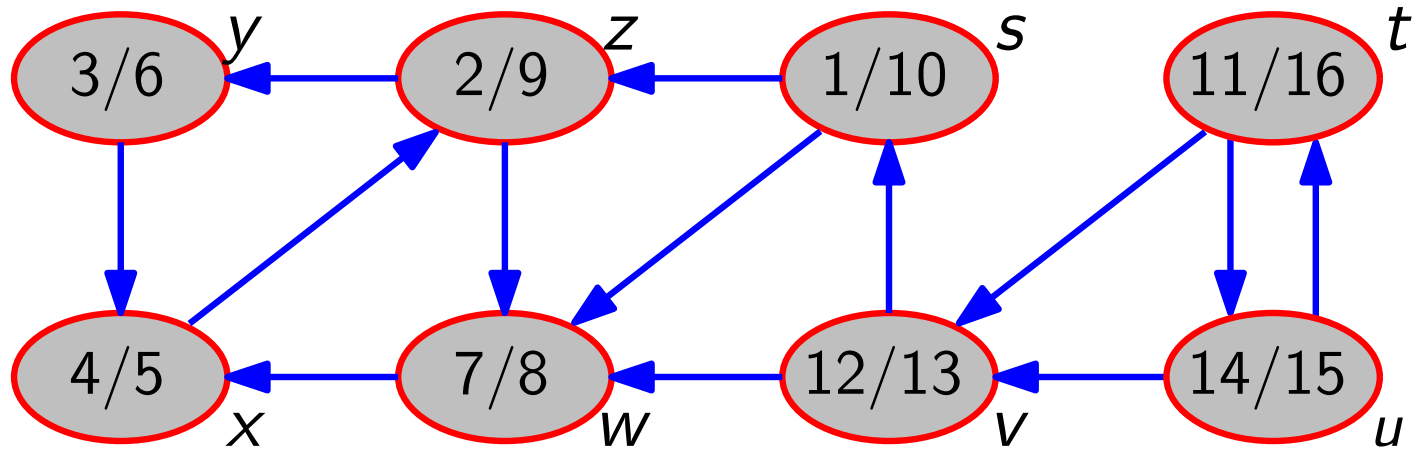
# Tiefensuche – Eigenschaften



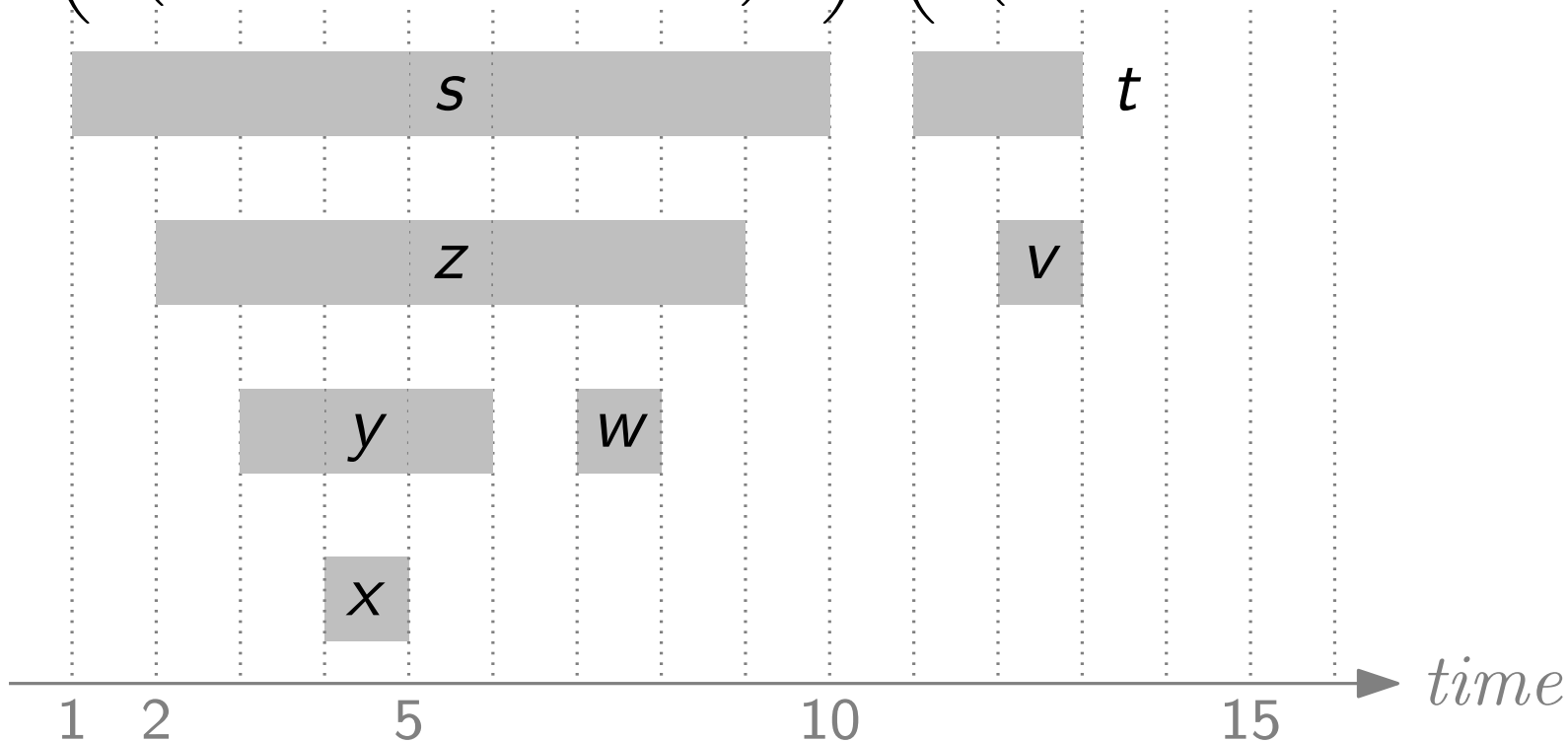
$(s (z (y (x x) y) (w w) z) s) (t (v$



# Tiefensuche – Eigenschaften

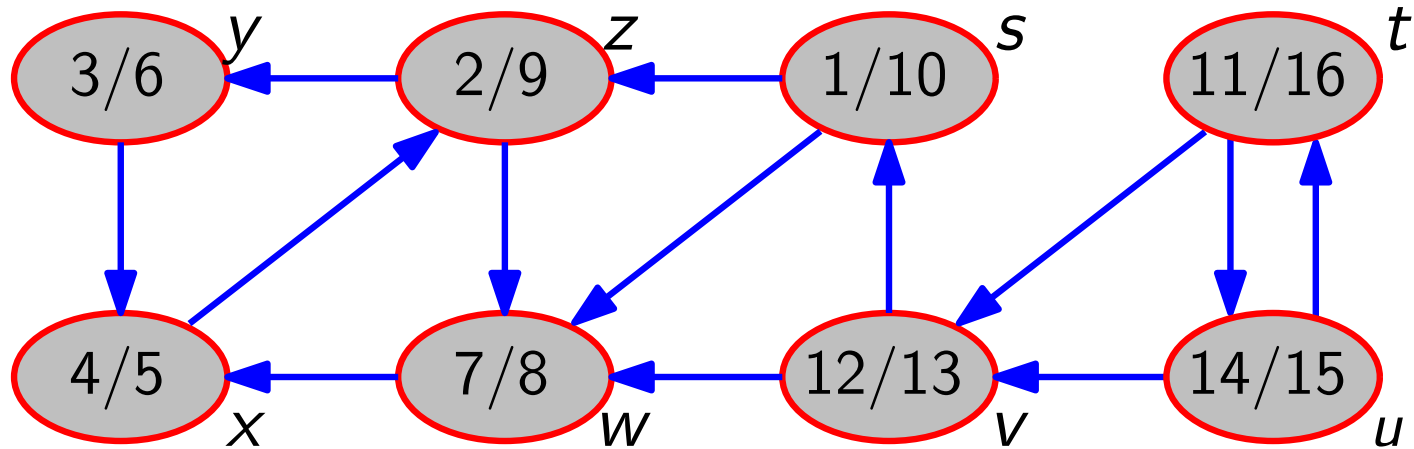


$(s (z (y (x x) y) (w w) z) s) (t (v$

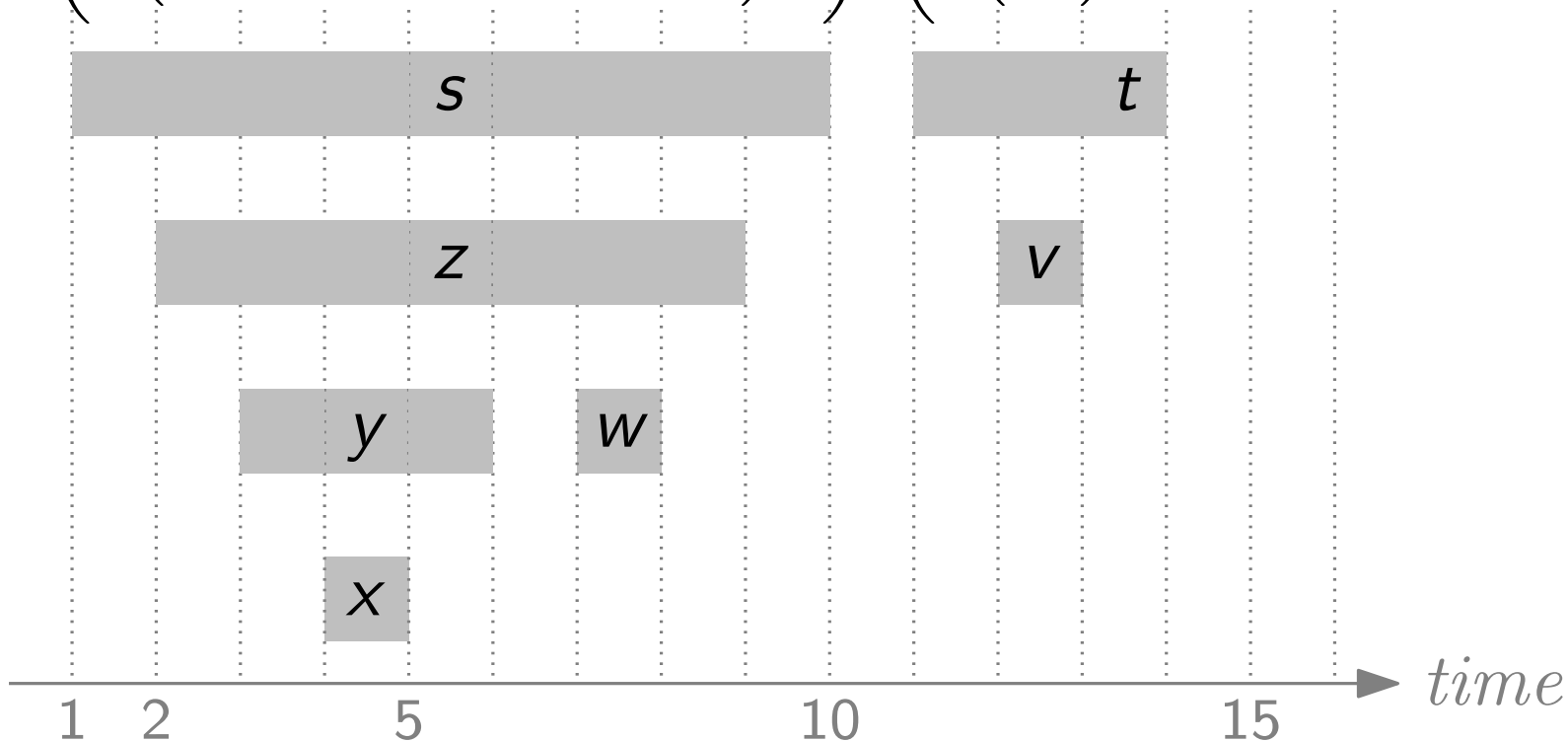




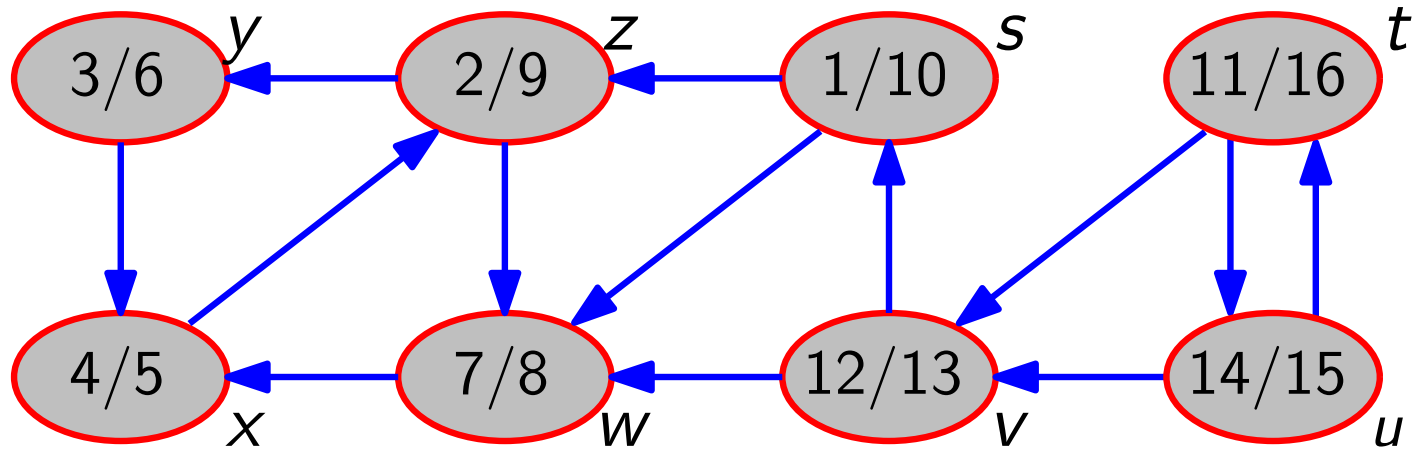
# Tiefensuche – Eigenschaften



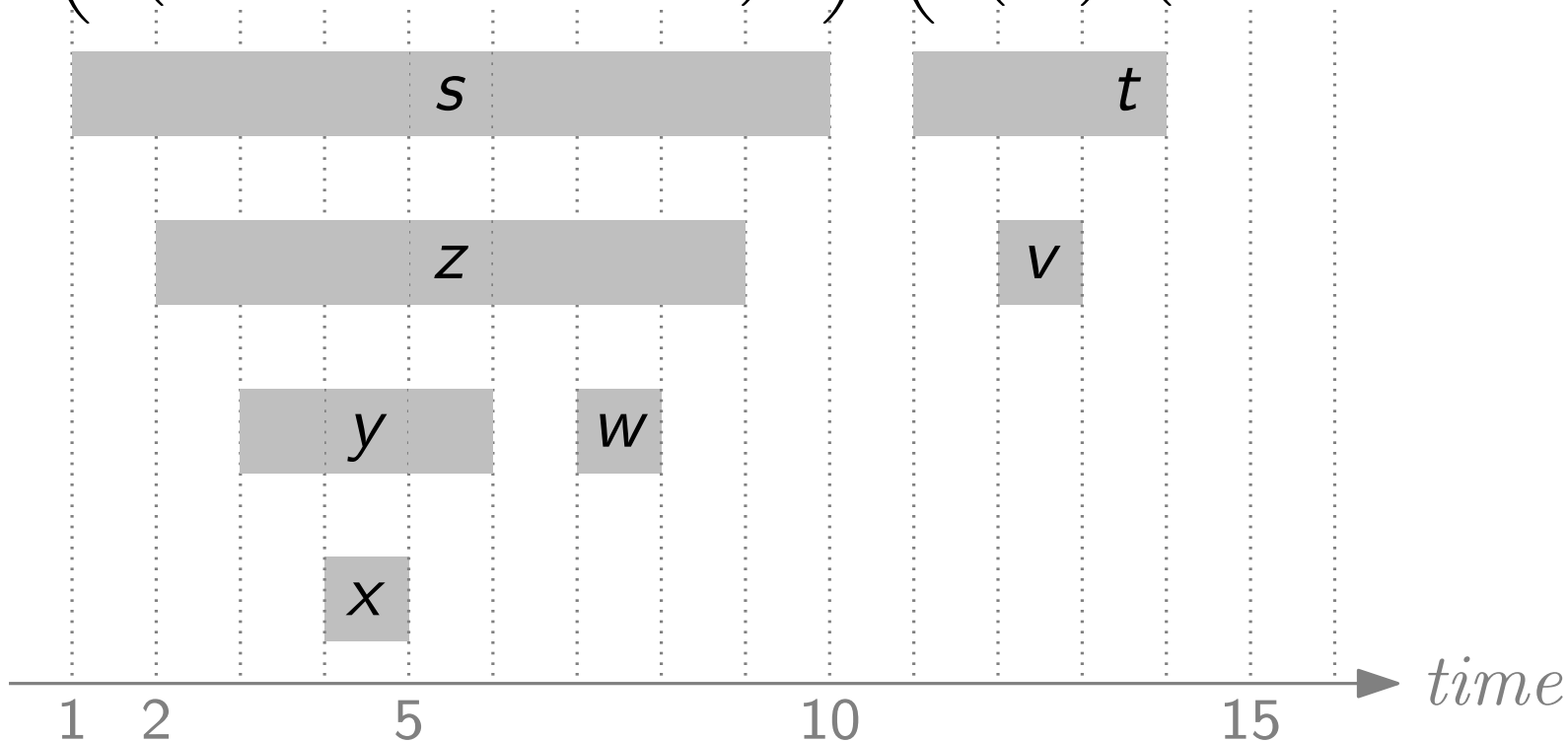
$(s (z (y (x x) y) (w w) z) s) (t (v v))$



# Tiefensuche – Eigenschaften



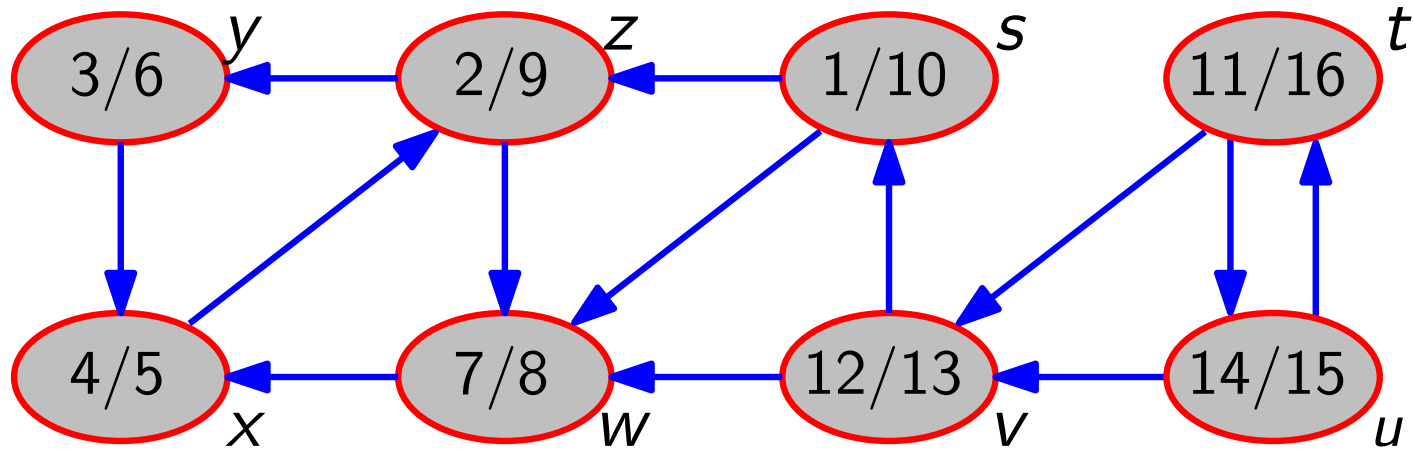
$(s (z (y (x x) y) (w w) z) s) (t (v v) (u$



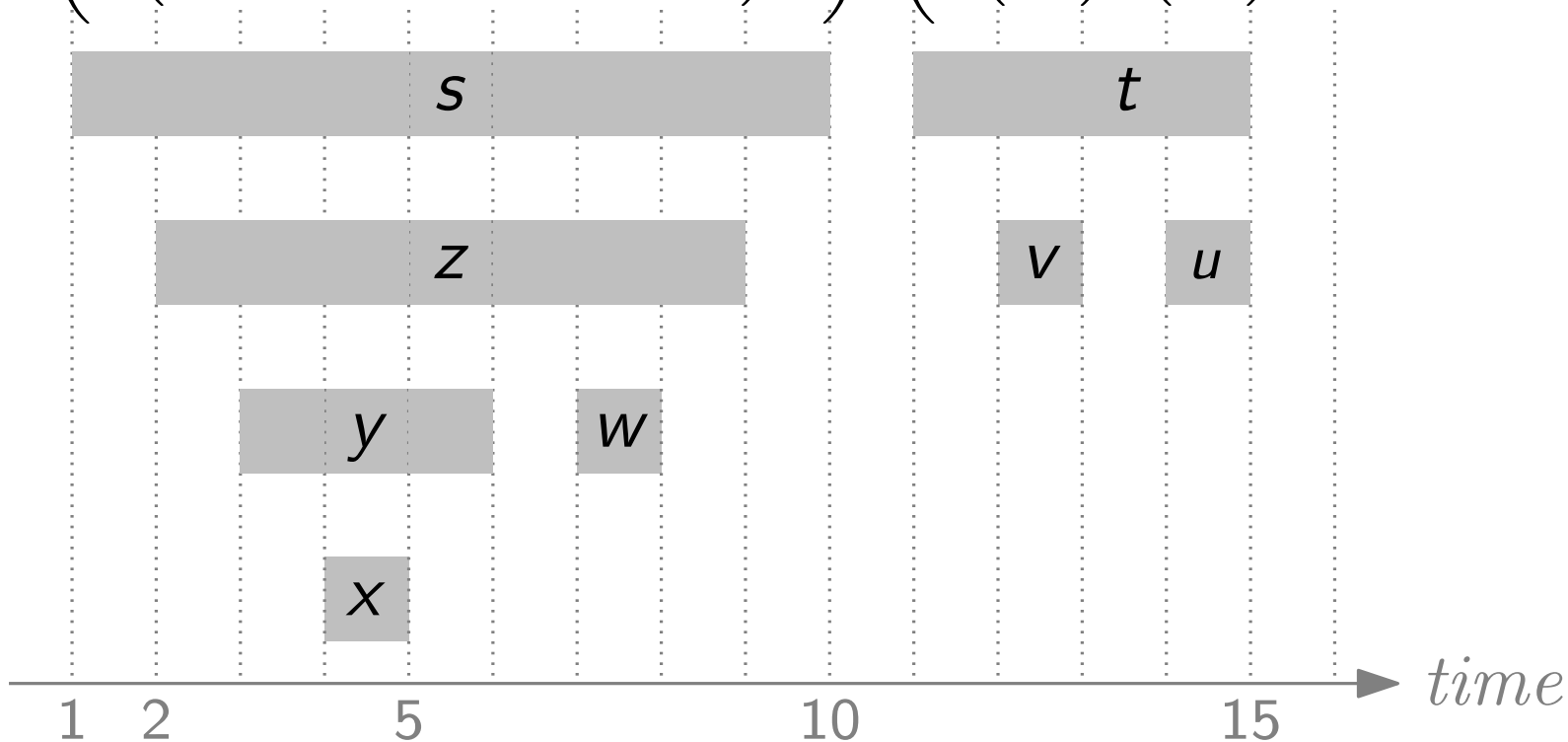




# Tiefensuche – Eigenschaften

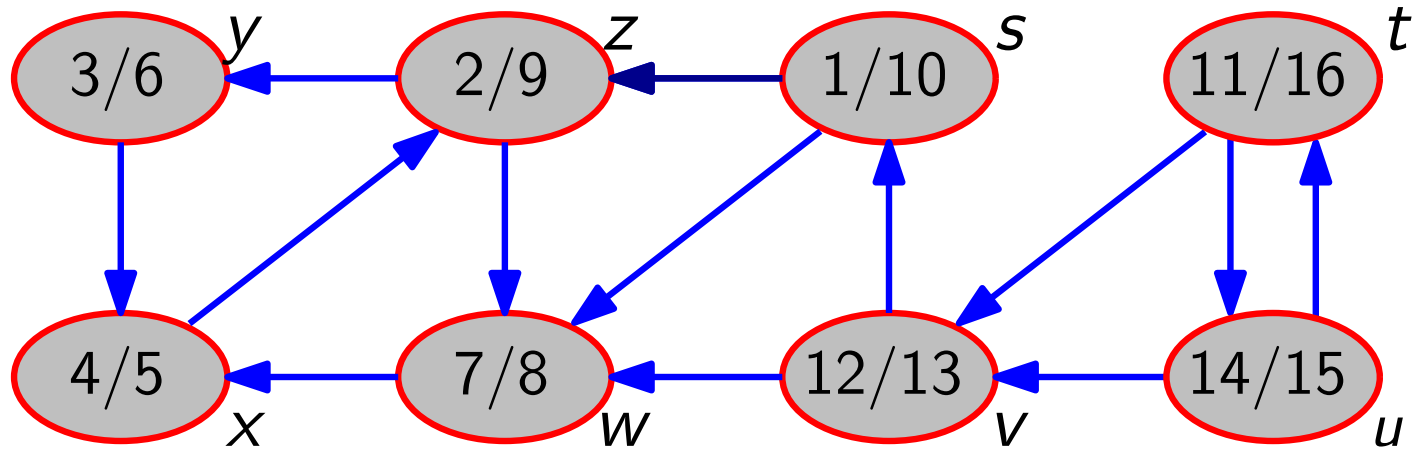


$(s (z (y (x x) y) (w w) z) s) (t (v v) (u u))$

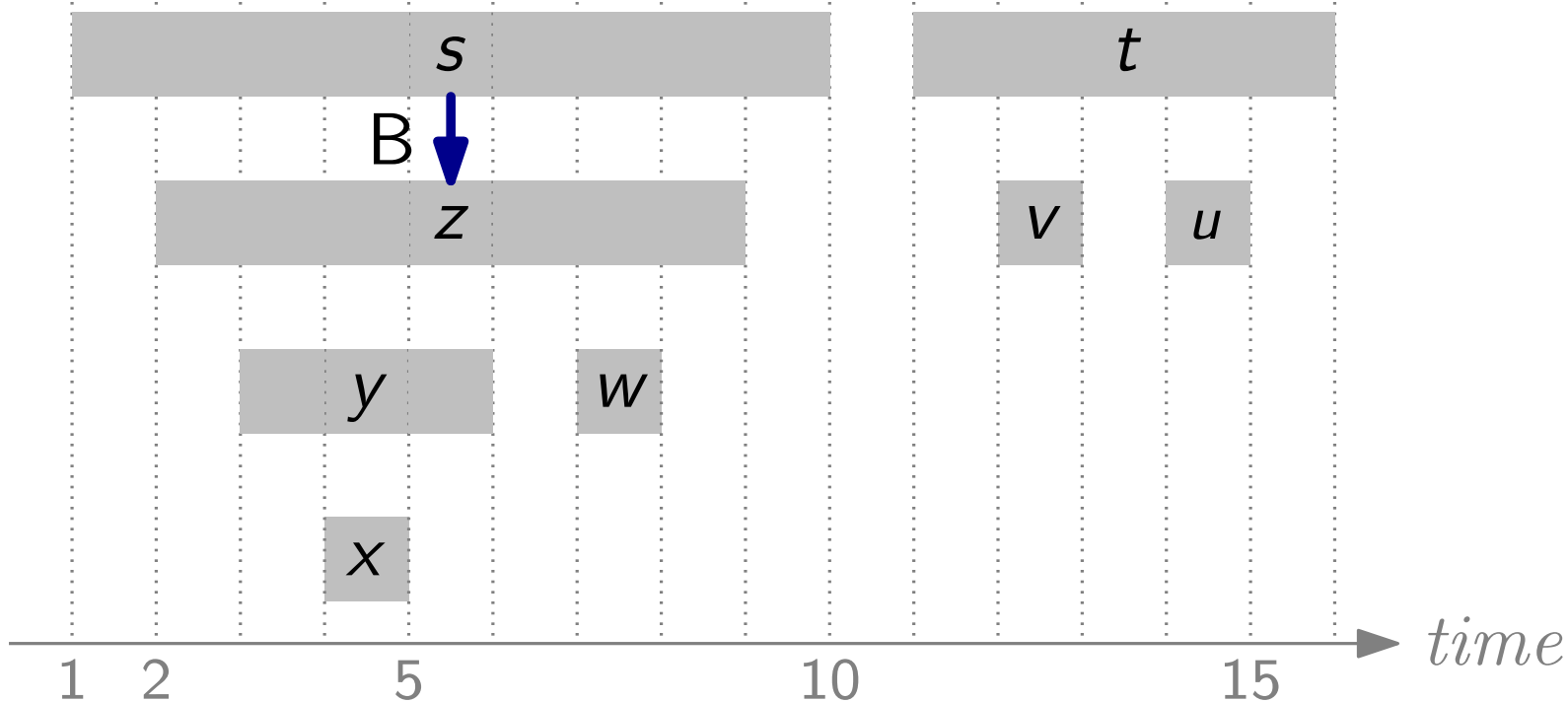




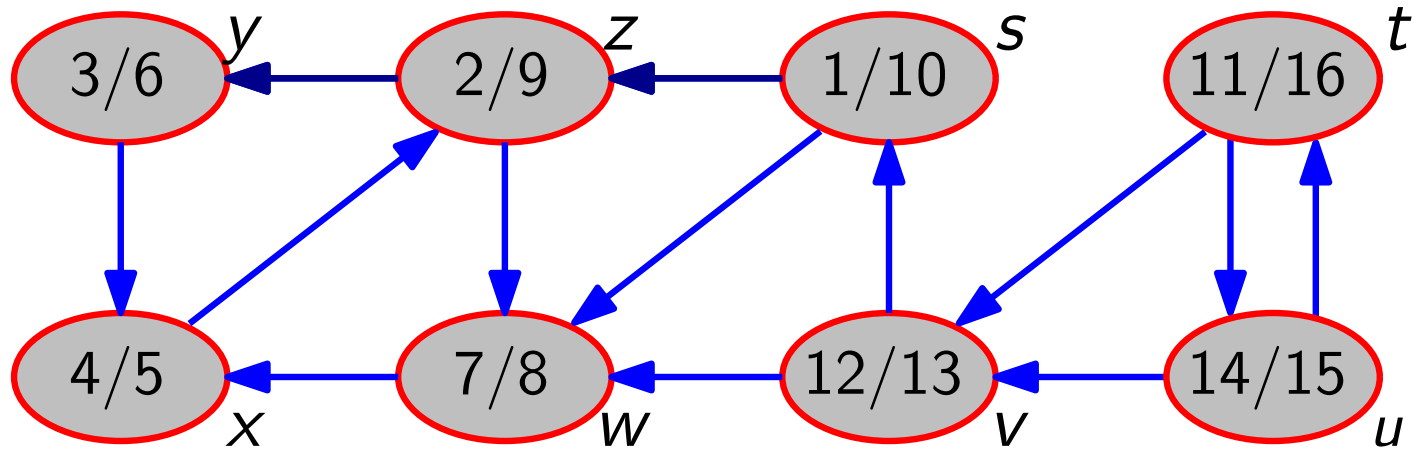
# Tiefensuche – Eigenschaften



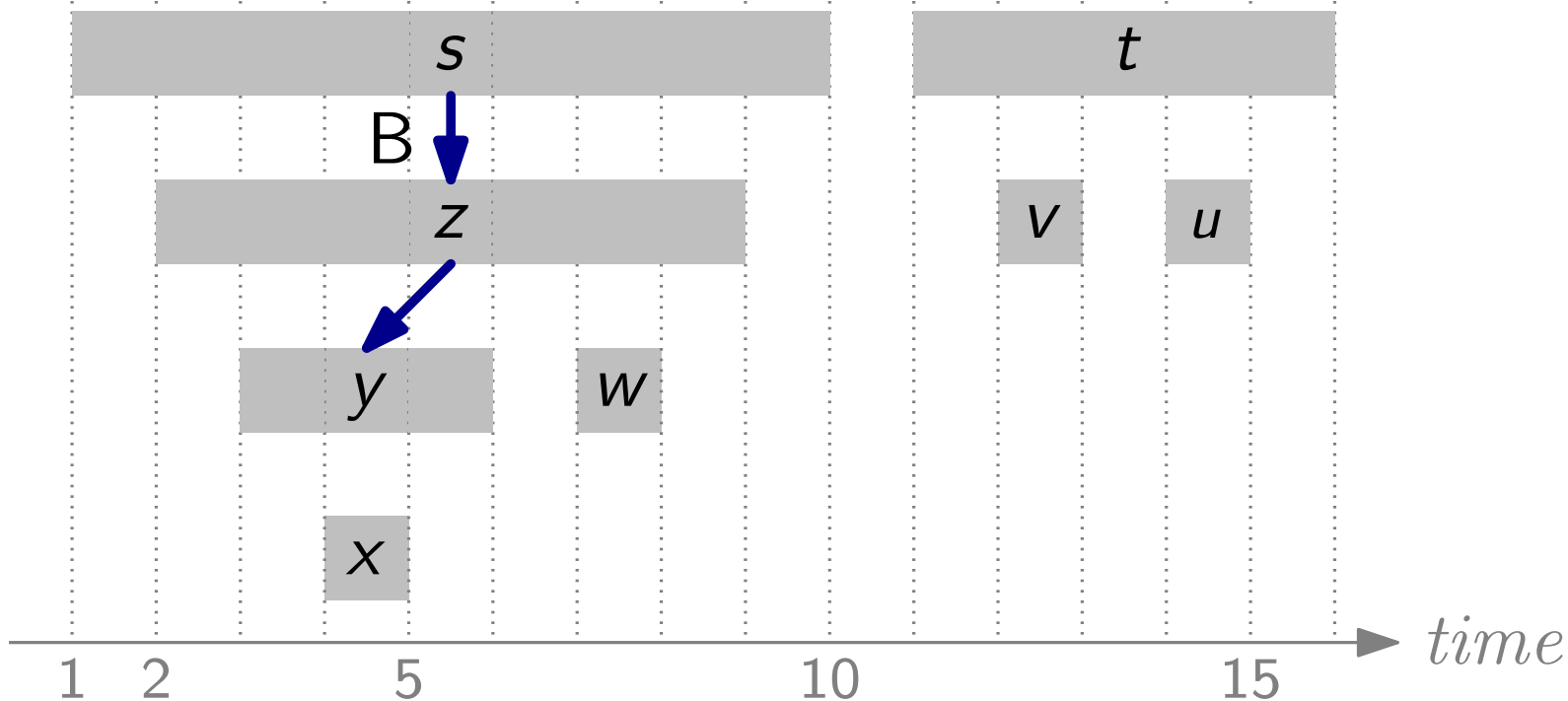
$(s (z (y (x x) y) (w w) z) s) (t (v v) (u u) t)$



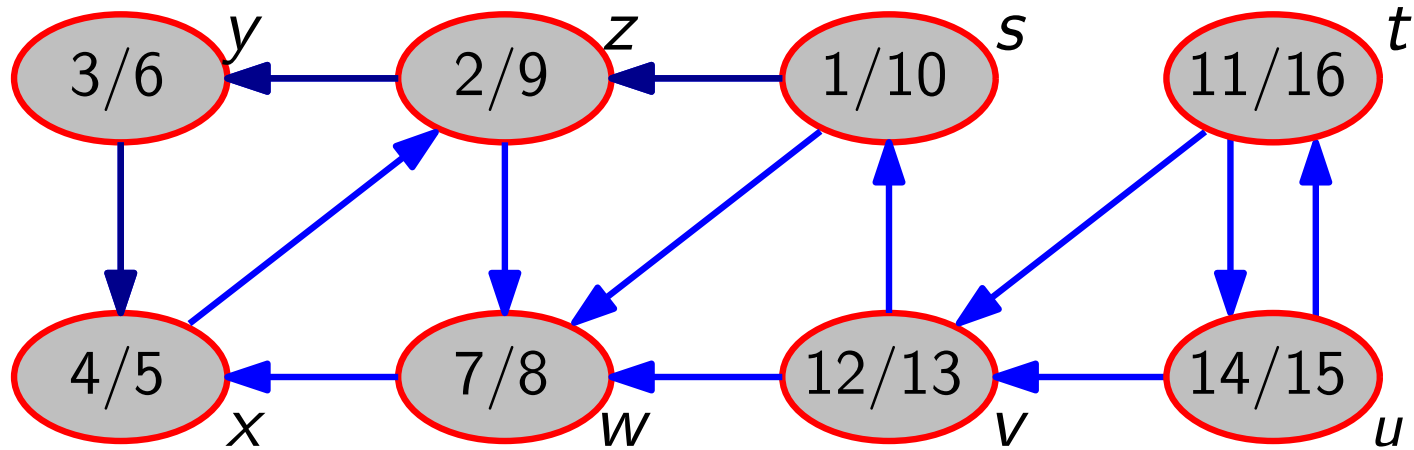
# Tiefensuche – Eigenschaften



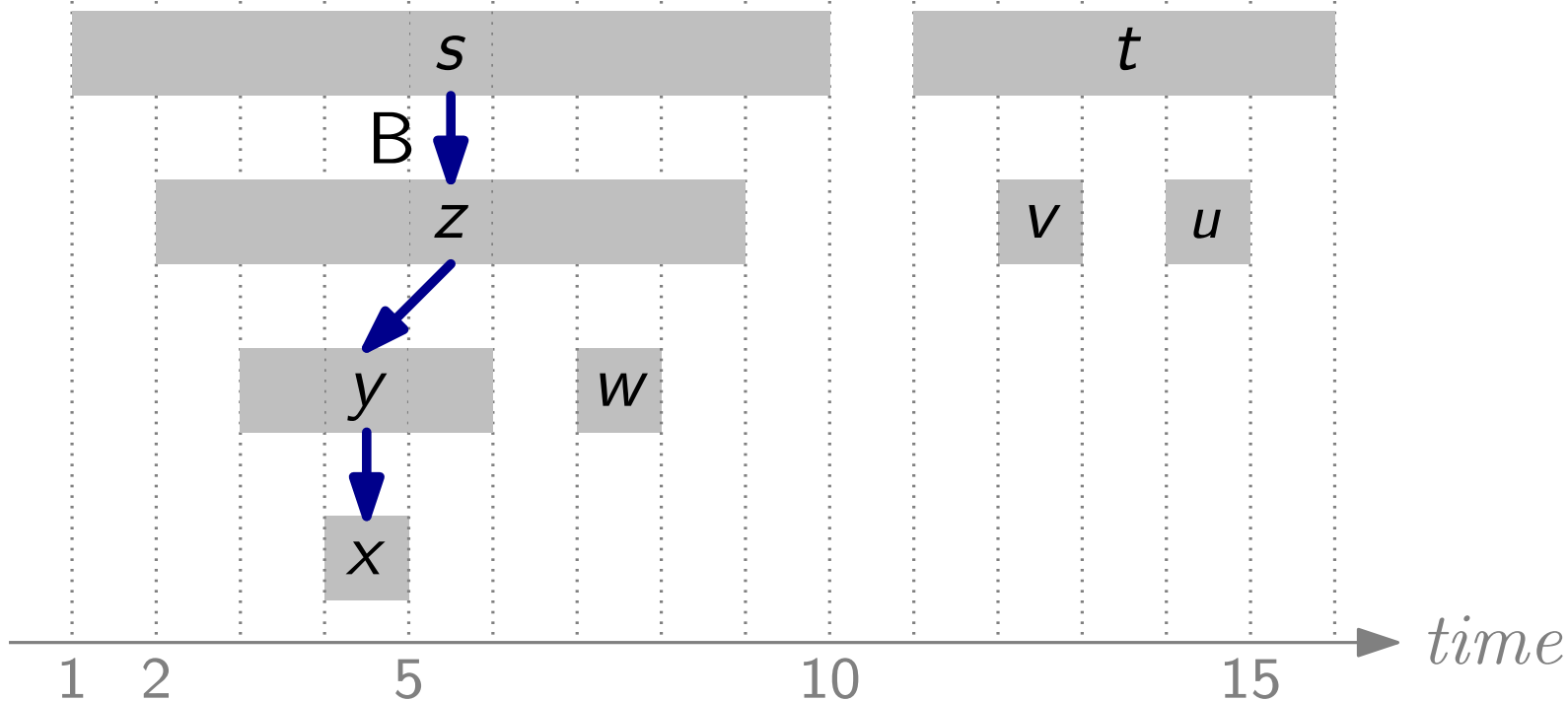
$(s (z (y (x x) y) (w w) z) s) (t (v v) (u u) t)$



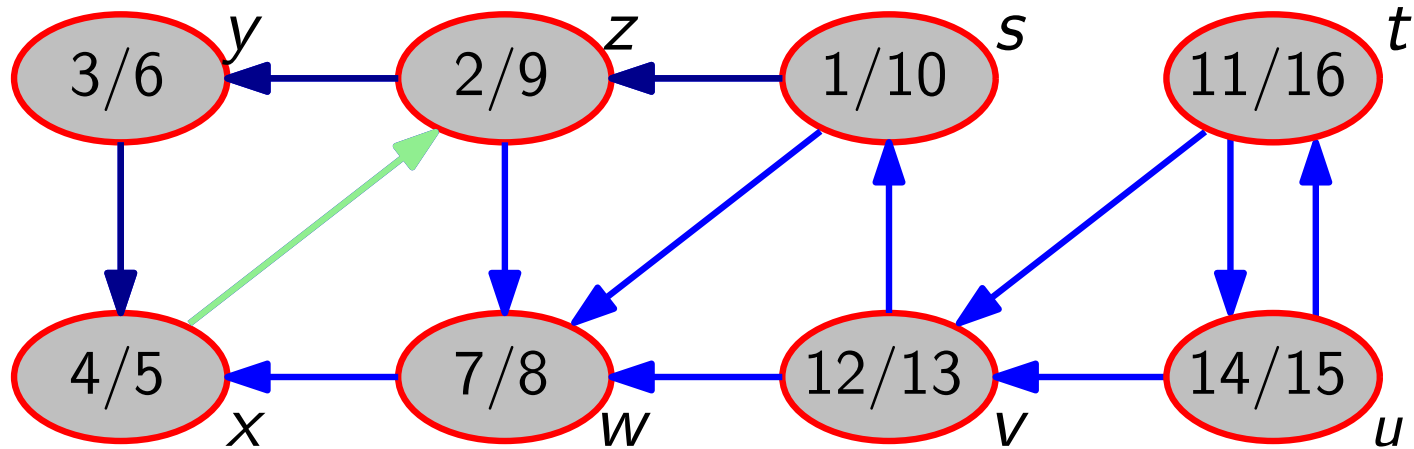
# Tiefensuche – Eigenschaften



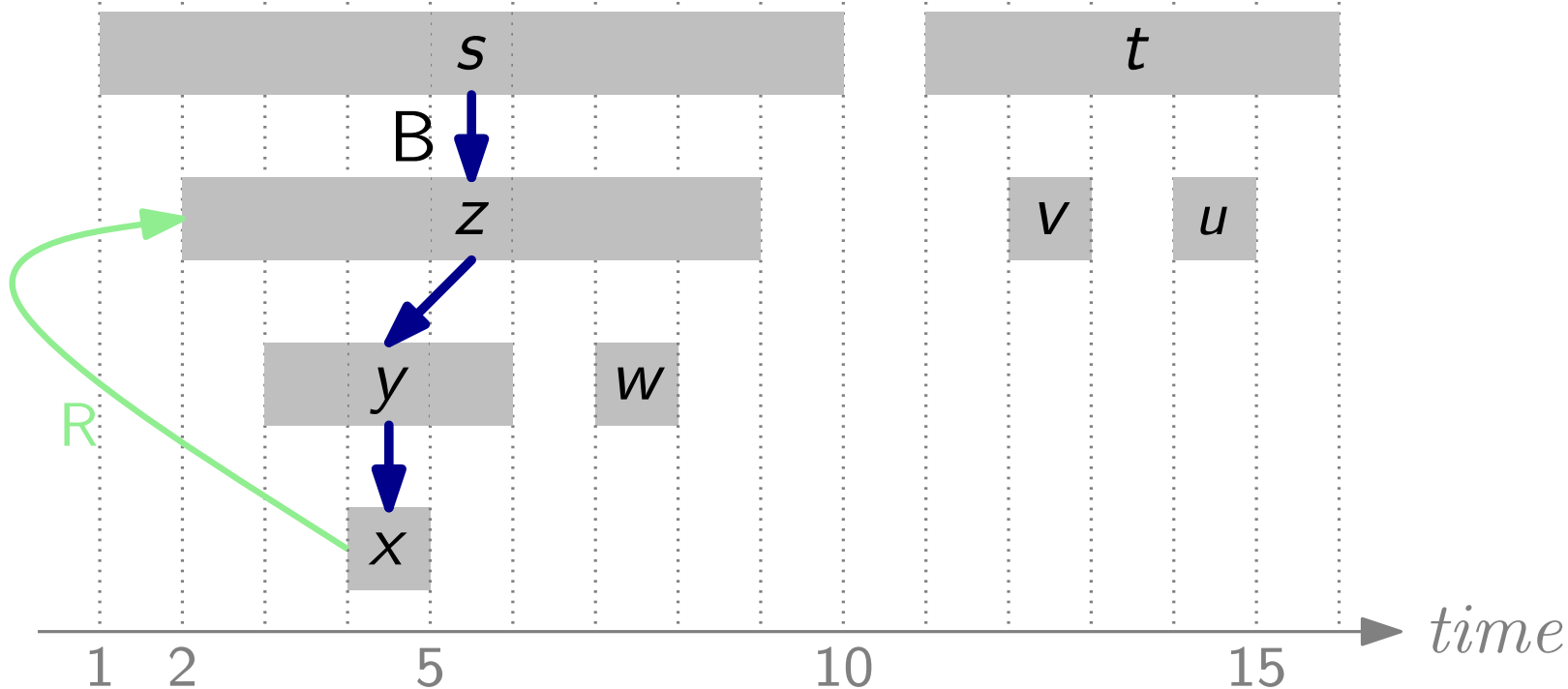
$(s (z (y (x x) y) (w w) z) s) (t (v v) (u u) t)$



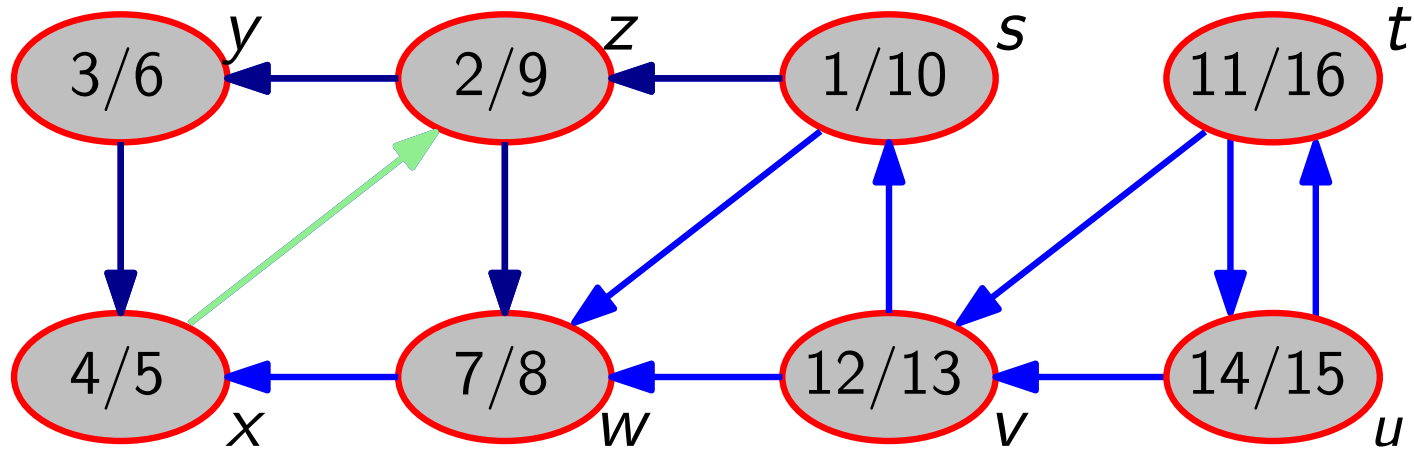
# Tiefensuche – Eigenschaften



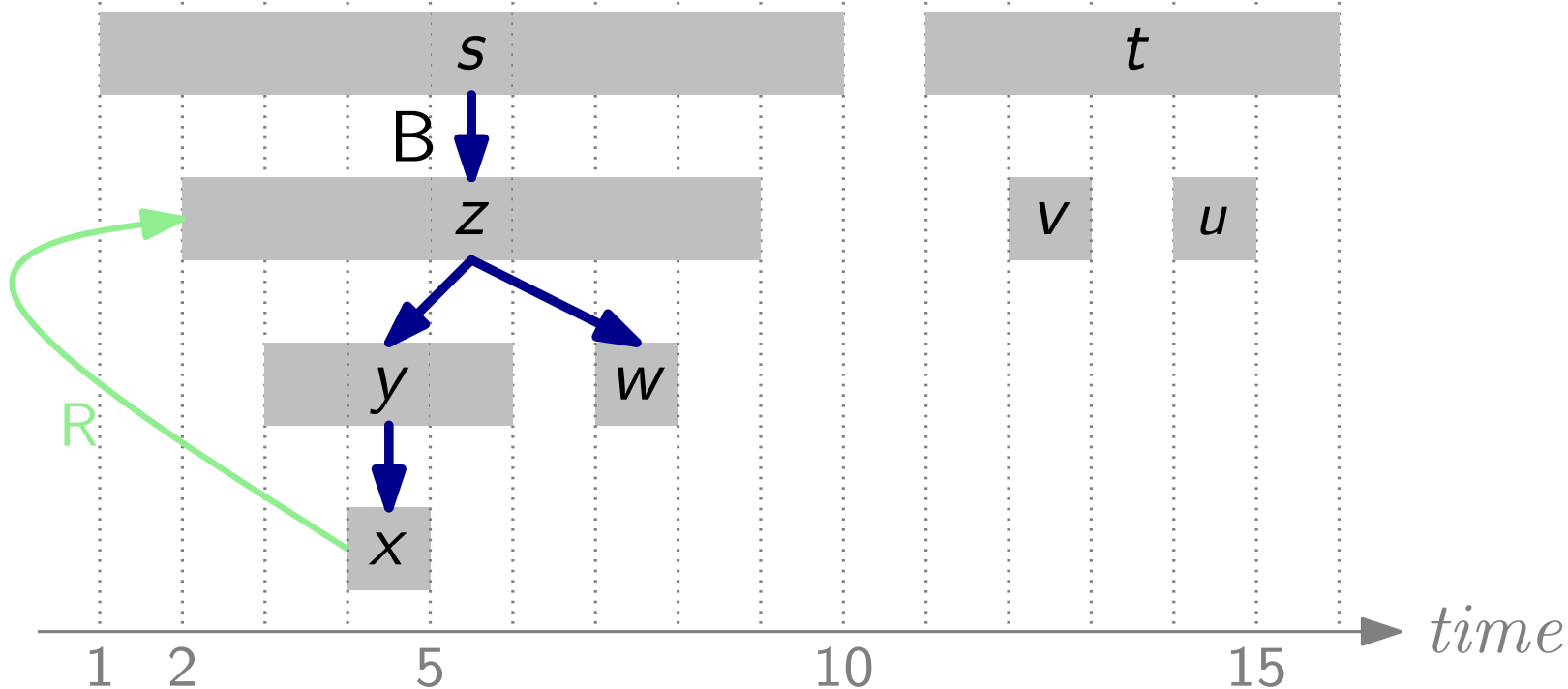
$(s (z (y (x x) y) (w w) z) s) (t (v v) (u u) t)$



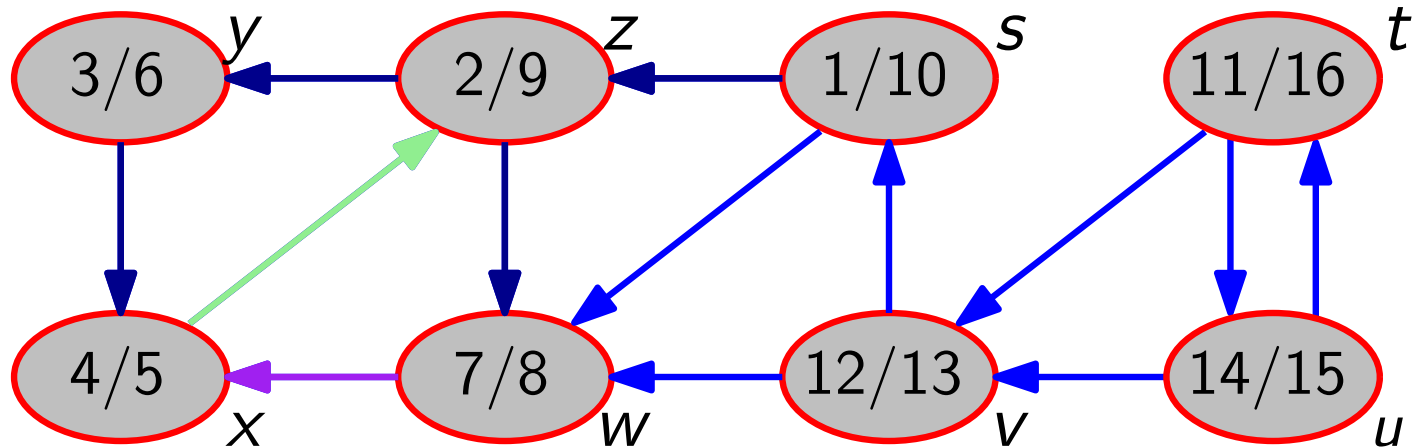
# Tiefensuche – Eigenschaften



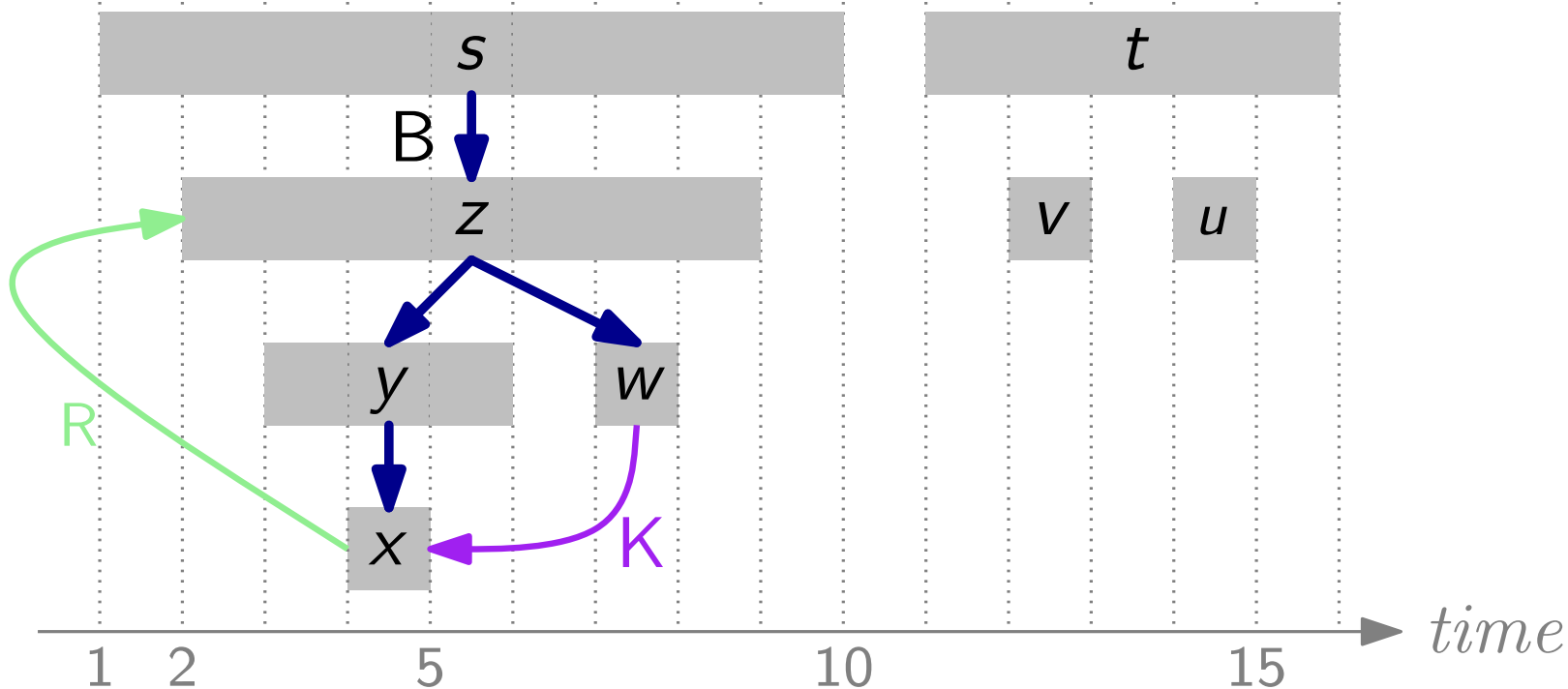
$$\left( s \left( z \left( y \left( x \ x \right) y \right) \left( w \ w \right) z \right) s \right) \quad \left( t \left( v \ v \right) \left( u \ u \right) t \right)$$



# Tiefensuche – Eigenschaften

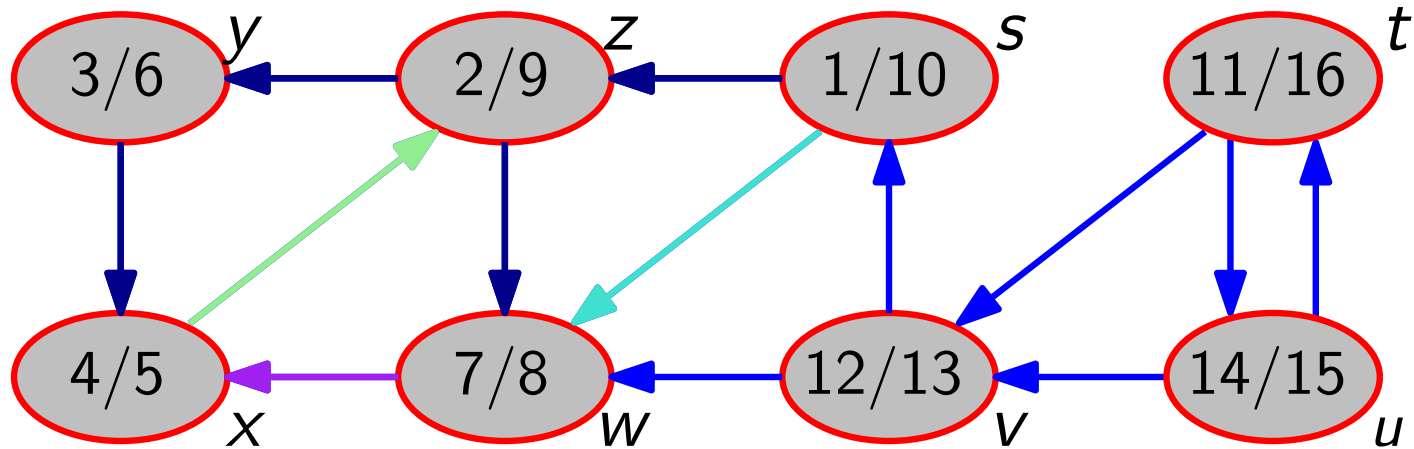


$$\left( s \left( z \left( y \left( x \ x \right) y \right) \left( w \ w \right) z \right) s \right) \quad \left( t \left( v \ v \right) \left( u \ u \right) t \right)$$

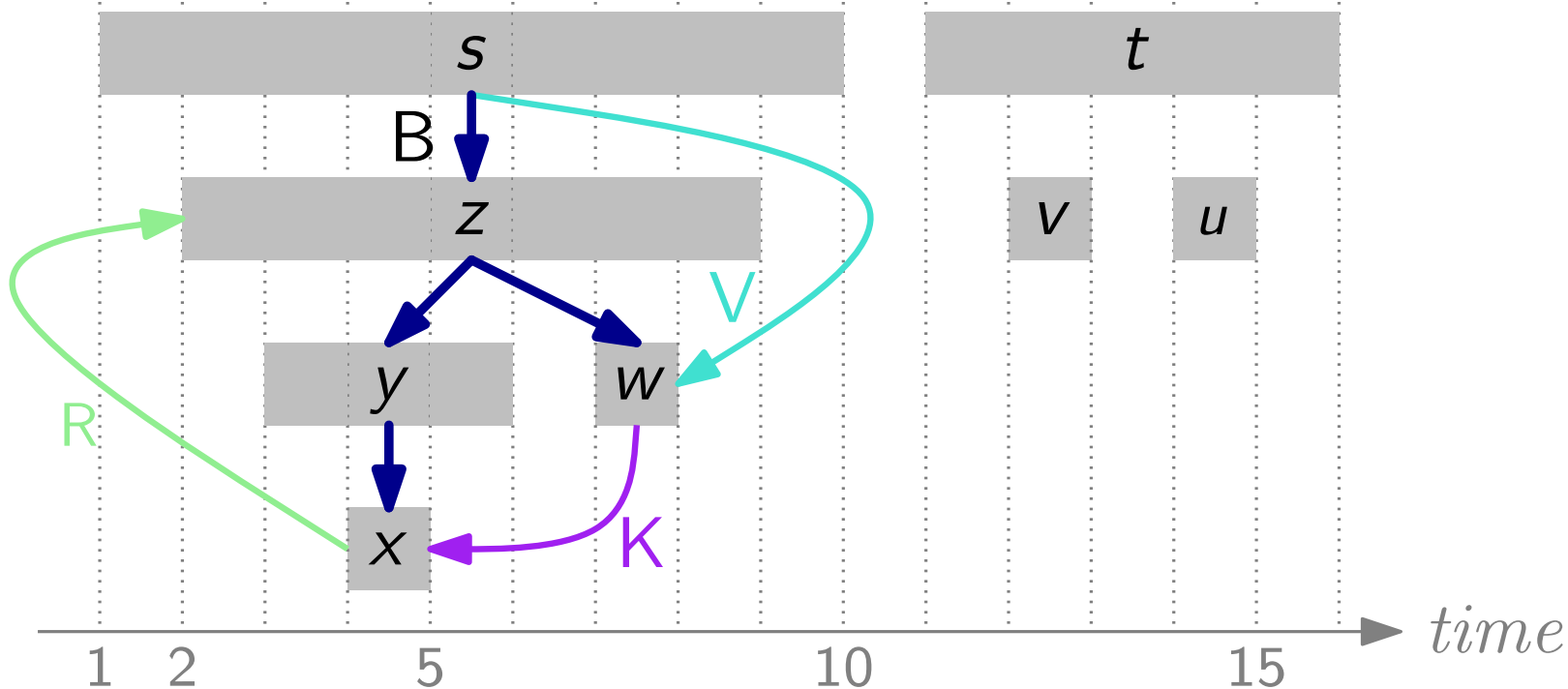




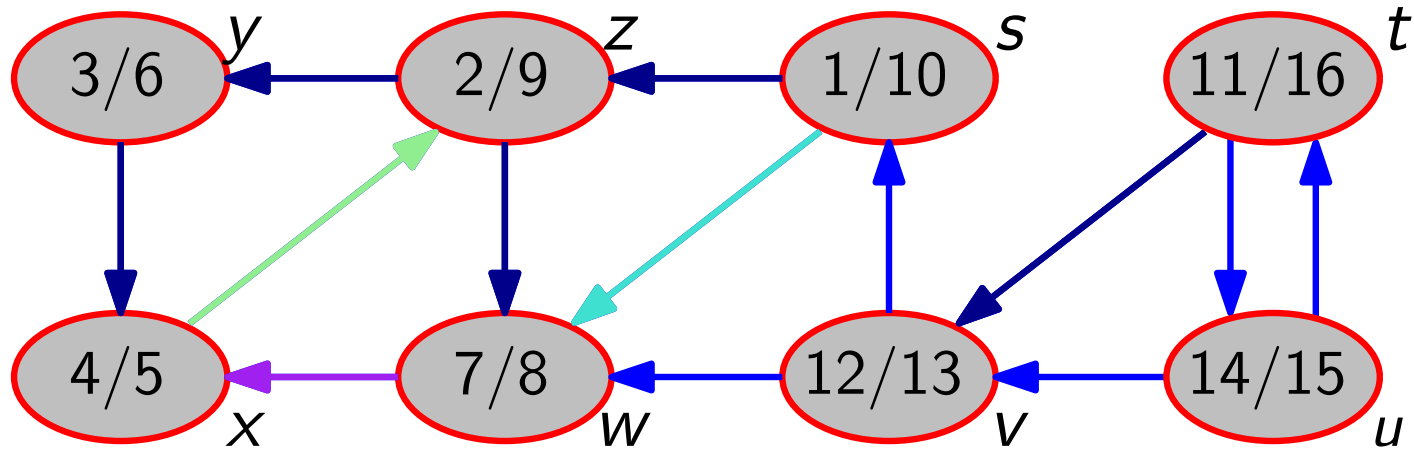
# Tiefensuche – Eigenschaften



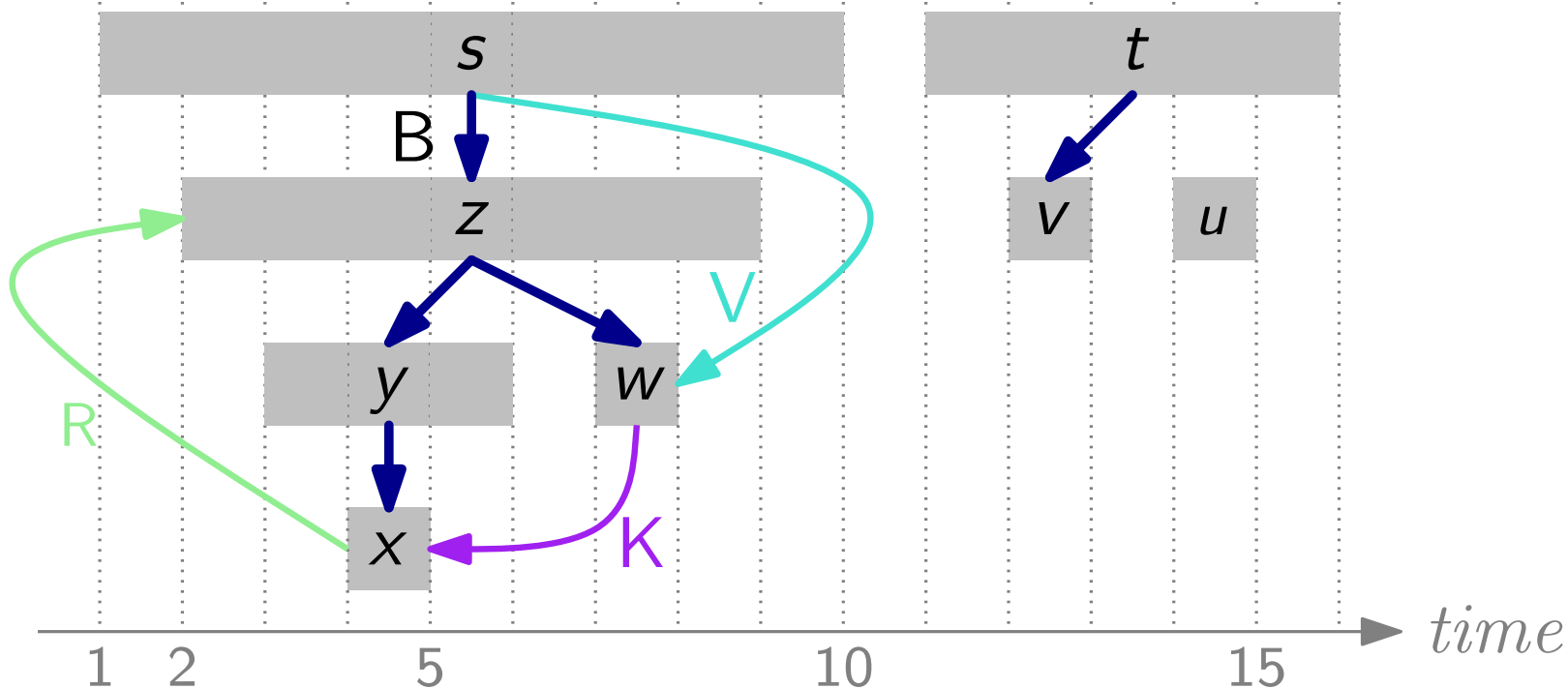
$(s (z (y (x x) y) (w w) z) s) \quad (t (v v) (u u) t)$



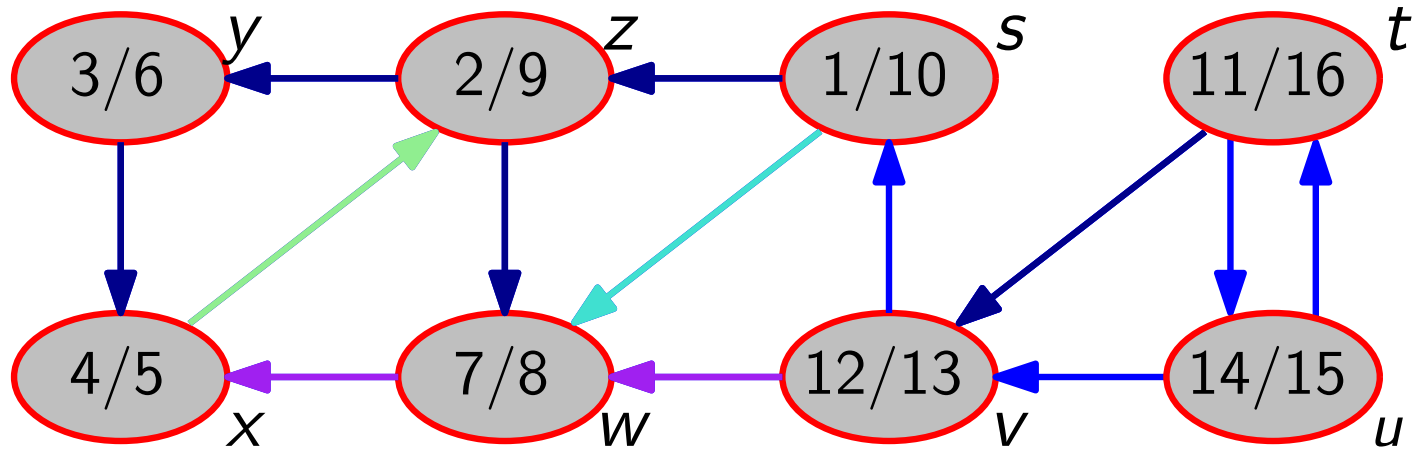
# Tiefensuche – Eigenschaften



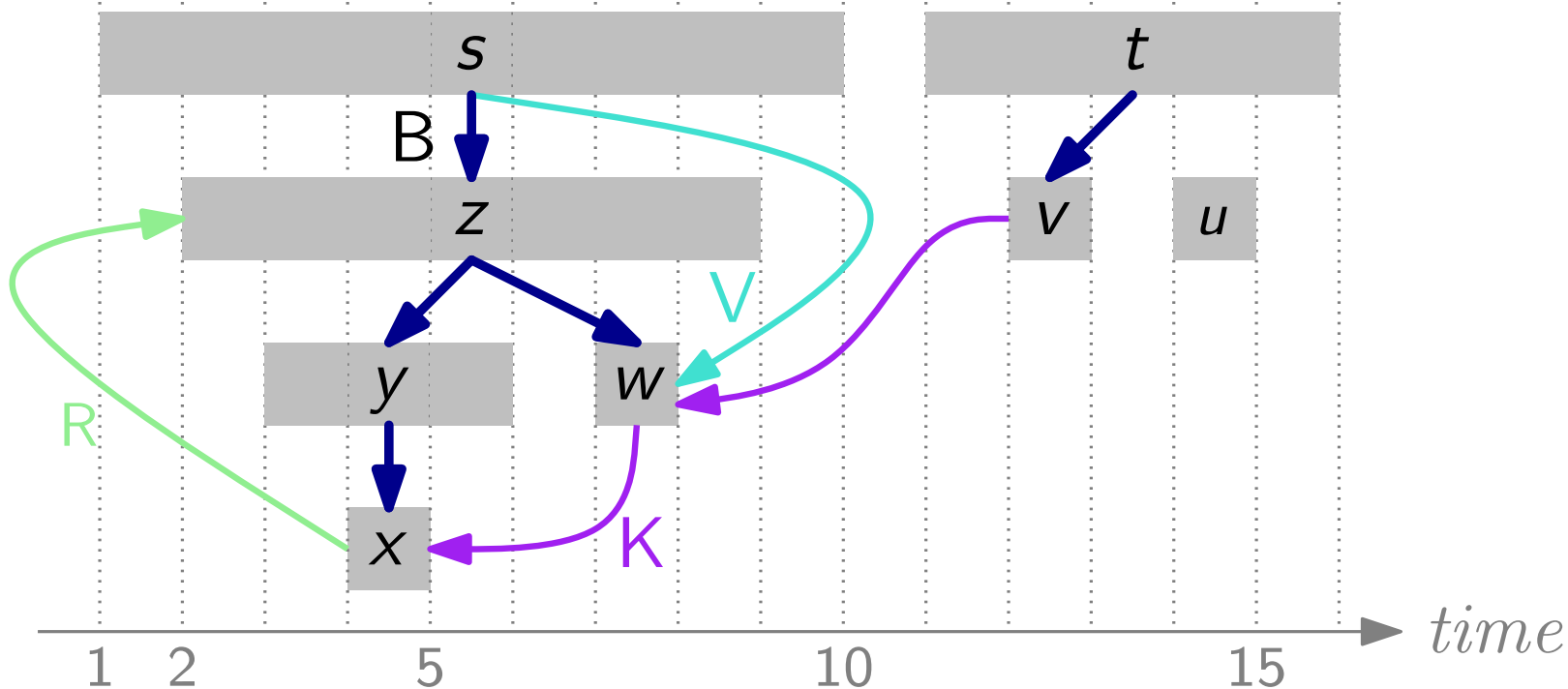
$$\left( s \left( z \left( y \left( x \ x \right) y \right) \left( w \ w \right) z \right) s \right) \quad \left( t \left( v \ v \right) \left( u \ u \right) t \right)$$



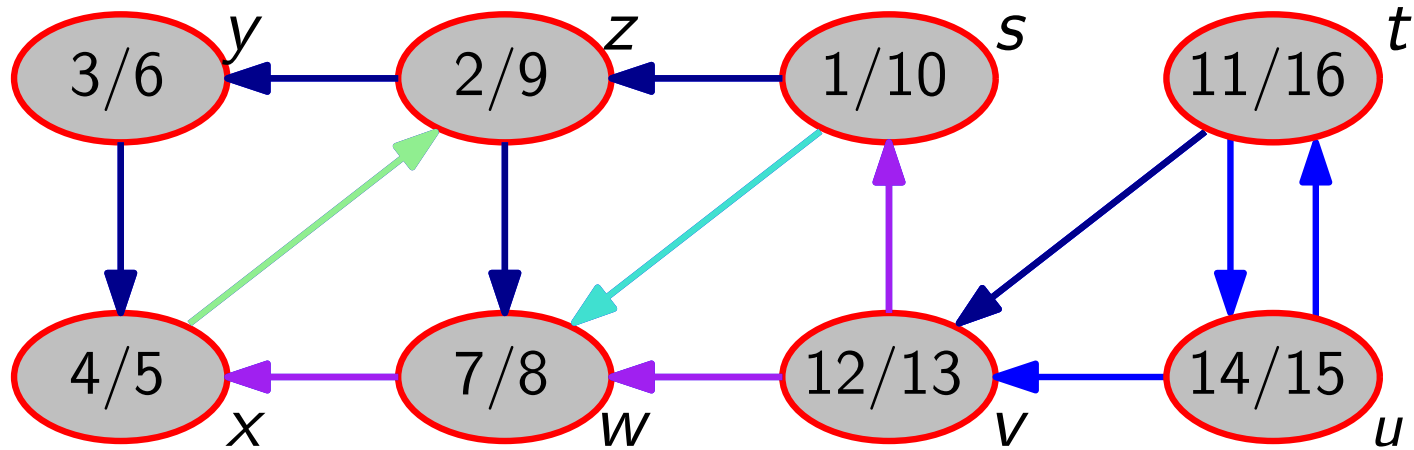
# Tiefensuche – Eigenschaften



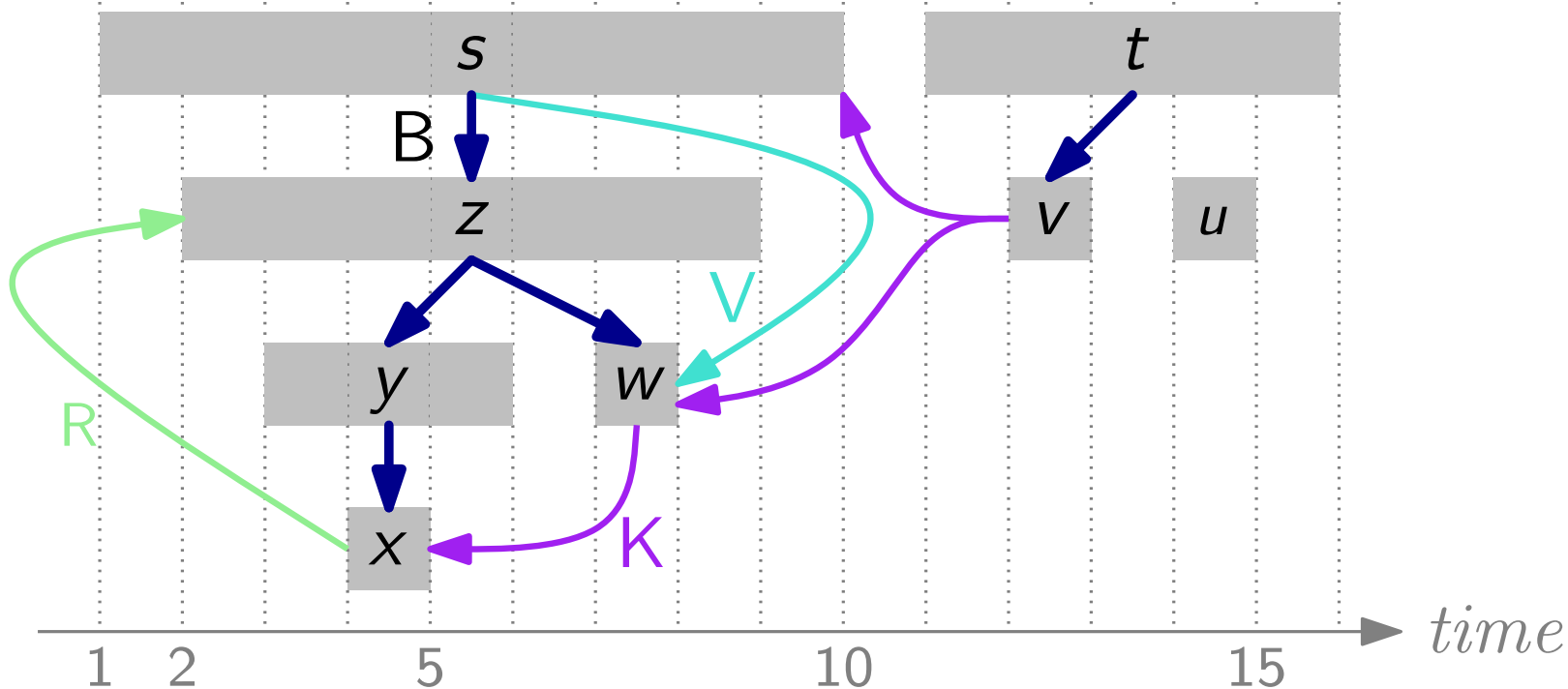
$$\left( s \left( z \left( y \left( x \ x \right) y \right) \left( w \ w \right) z \right) s \right) \left( t \left( v \ v \right) \left( u \ u \right) t \right)$$



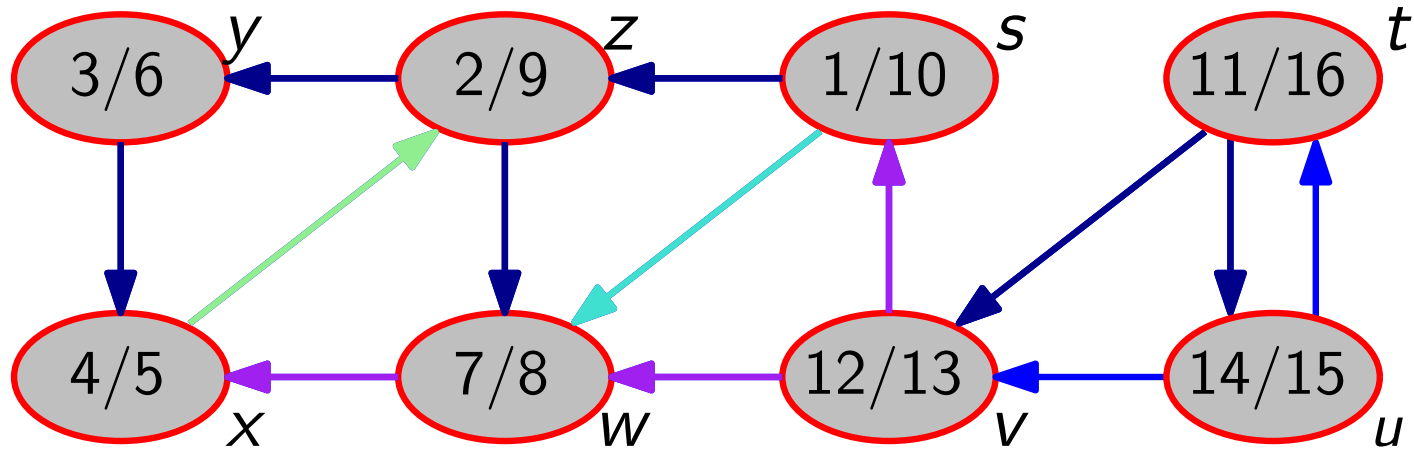
# Tiefensuche – Eigenschaften



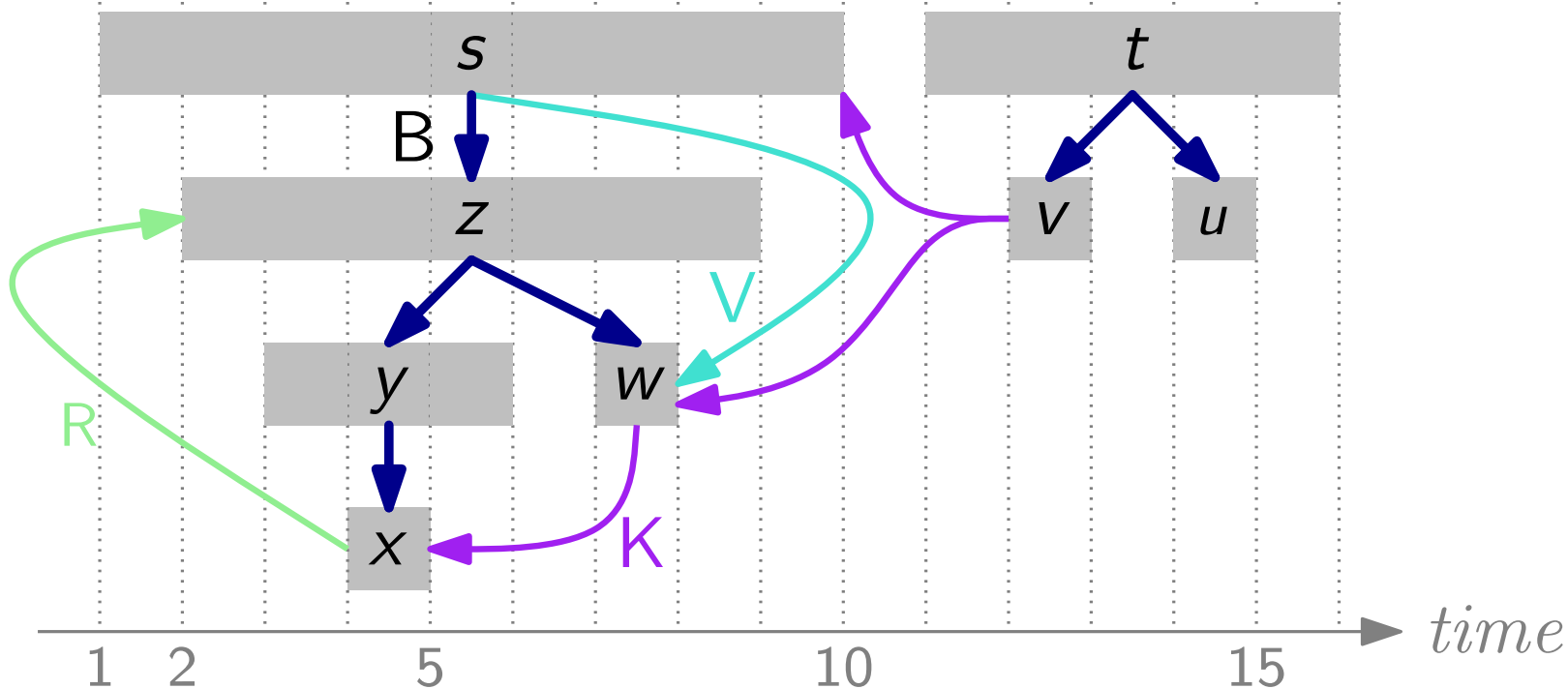
$(s (z (y (x x) y) (w w) z) s) (t (v v) (u u) t)$



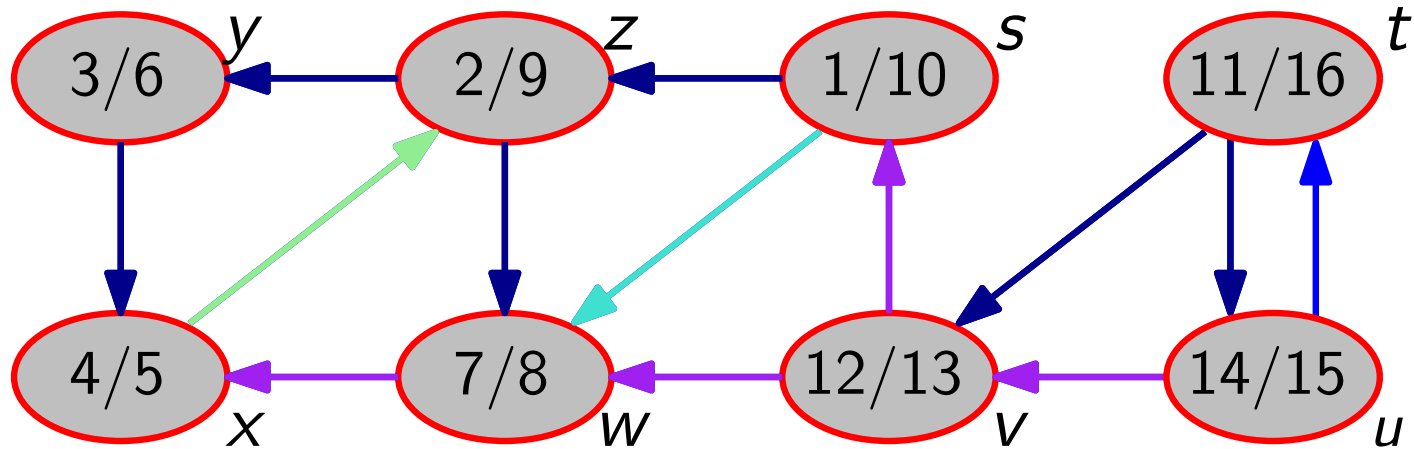
# Tiefensuche – Eigenschaften



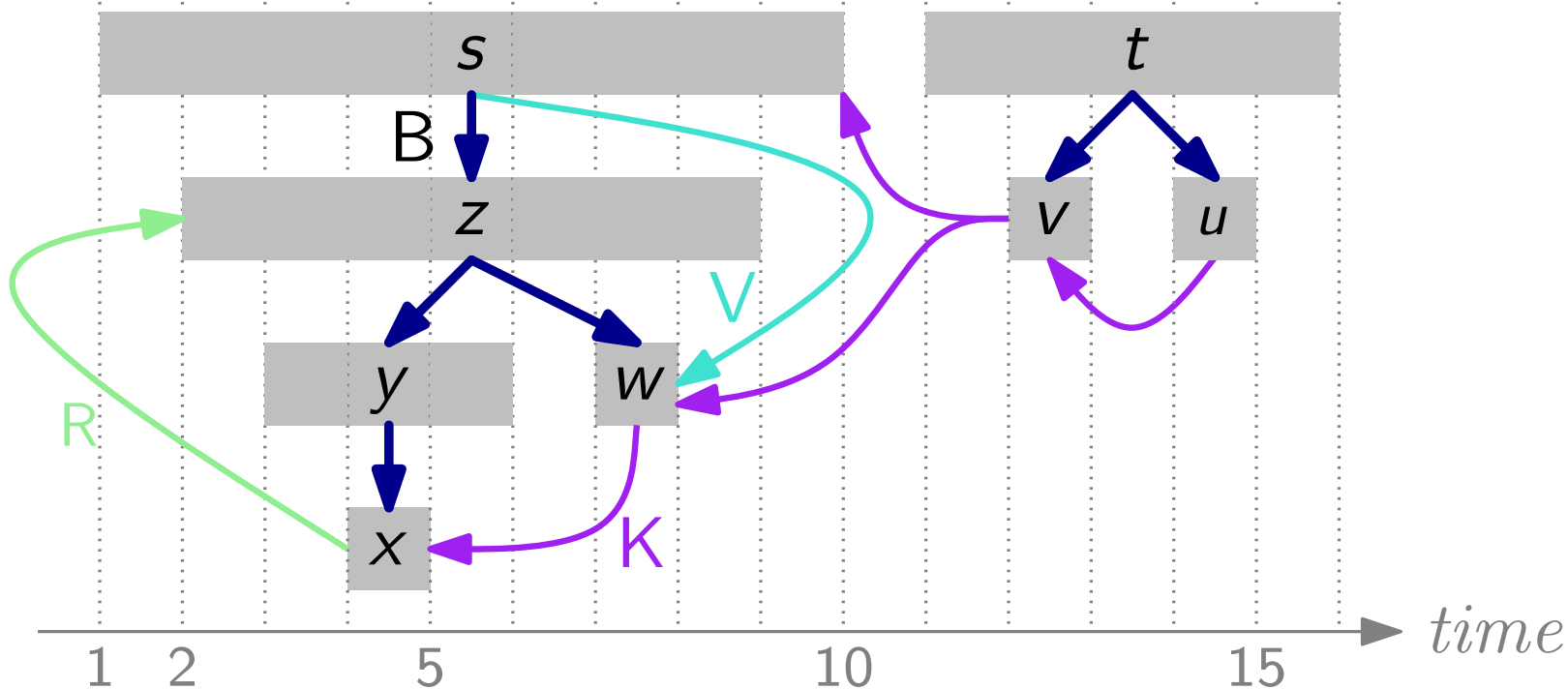
$(s (z (y (x x) y) (w w) z) s) (t (v v) (u u) t)$



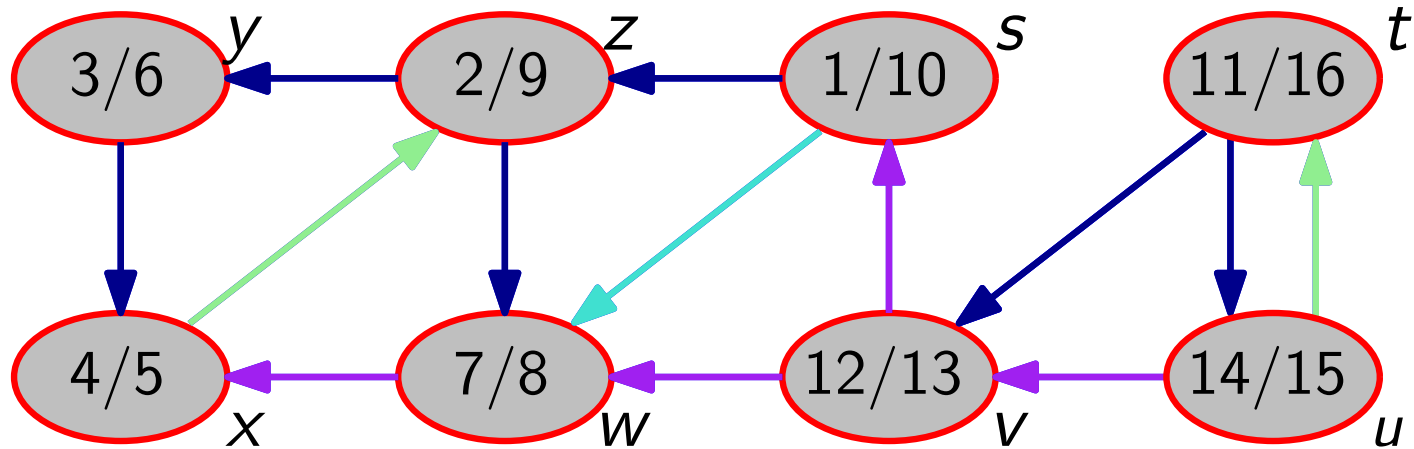
# Tiefensuche – Eigenschaften



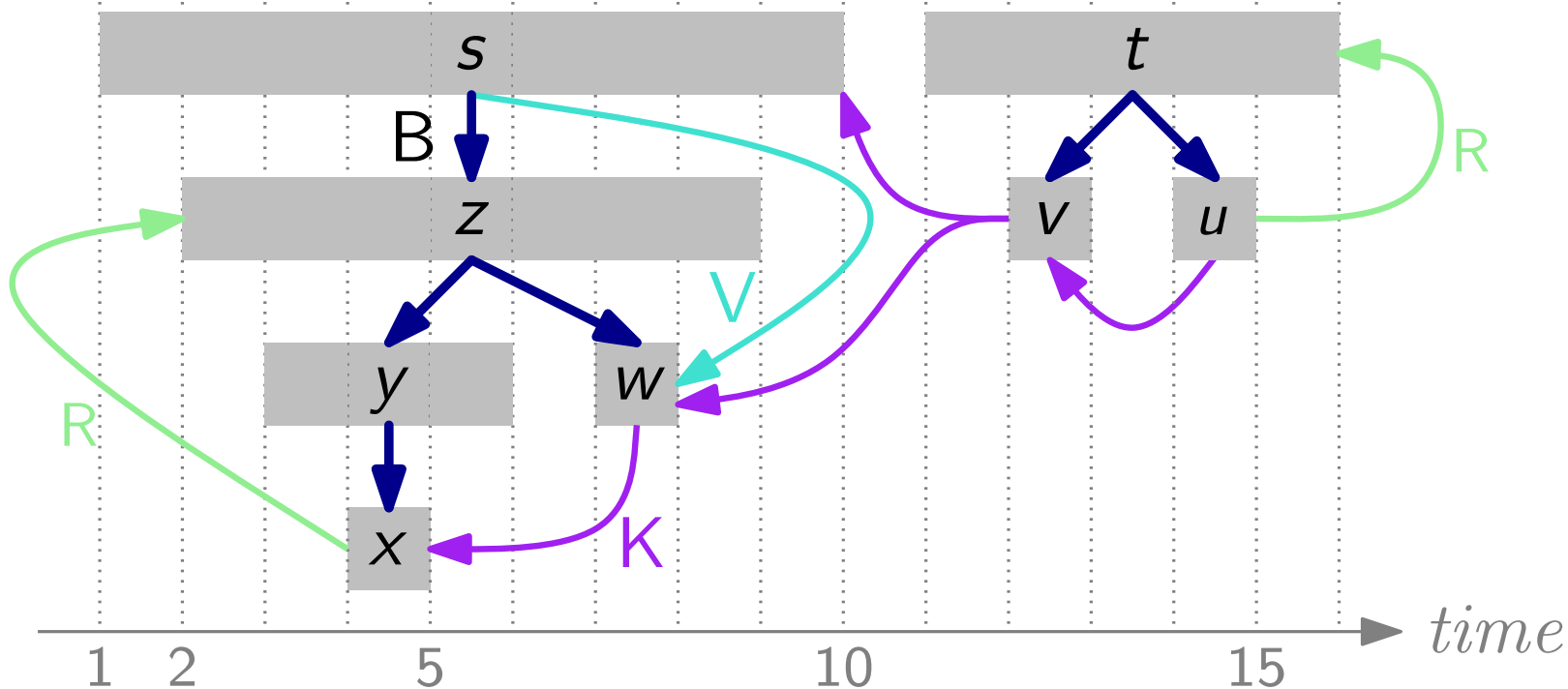
$$\left( s \left( z \left( y \left( x \ x \right) y \right) \left( w \ w \right) z \right) s \right) \left( t \left( v \ v \right) \left( u \ u \right) t \right)$$



# Tiefensuche – Eigenschaften



$(s (z (y (x x) y) (w w) z) s) (t (v v) (u u) t)$



# Tiefensuche – Analyse

## Satz.

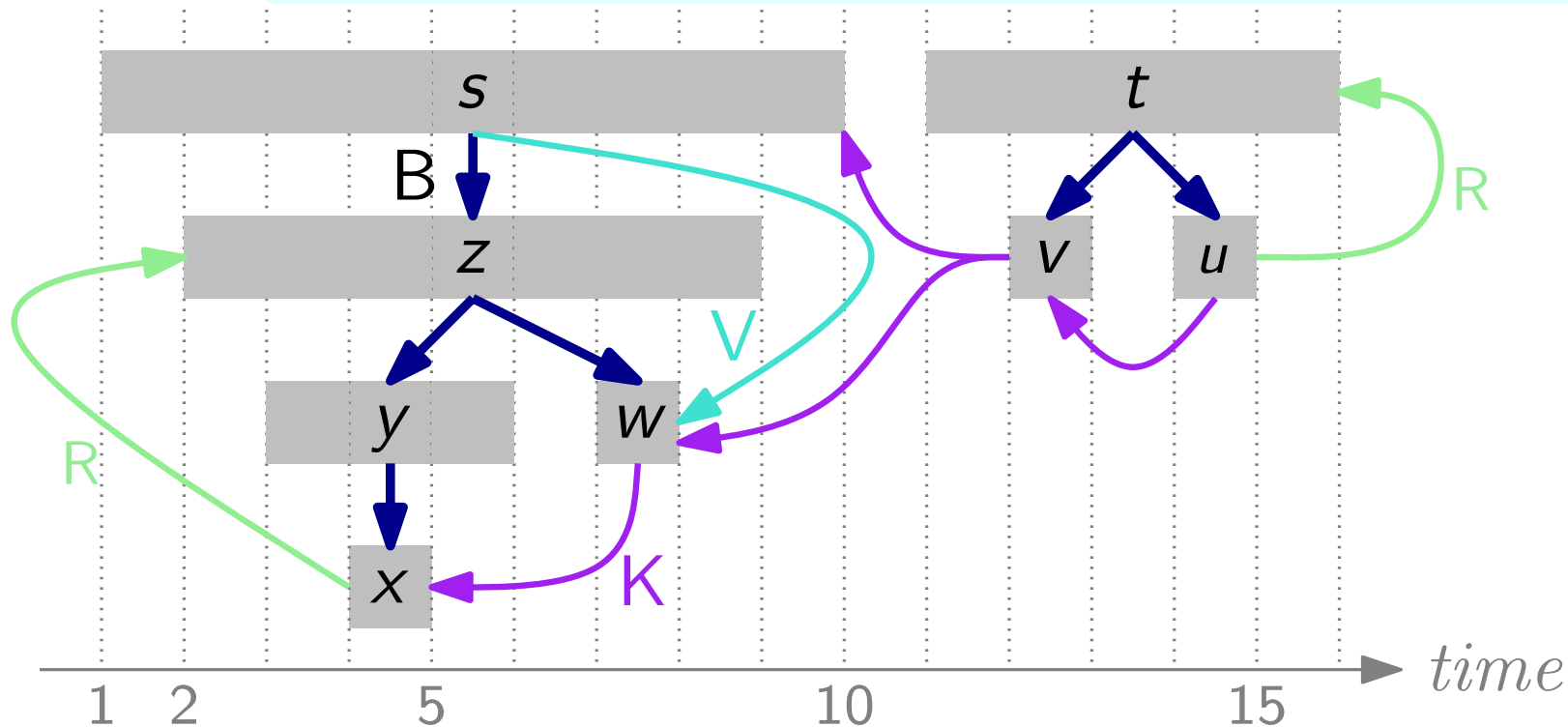
(Klammerentheorem)

Nach  $\text{DFS}(G)$  gilt für  $\{u, v\} \in \binom{V}{2}$  genau eine der Bedingungen

(i)

(ii)

(iii)





# Tiefensuche – Analyse

## Satz.

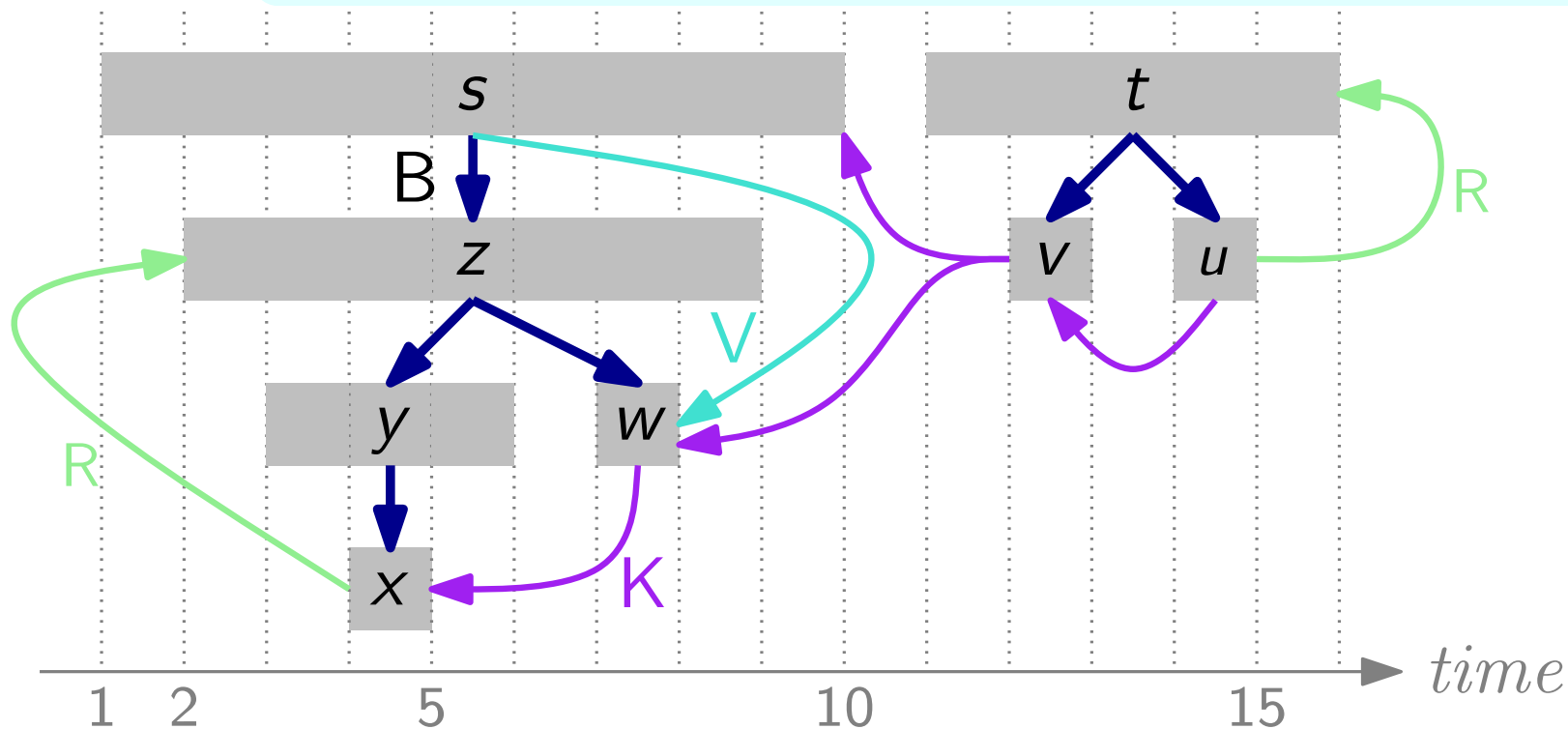
(Klammerentheorem)

Nach  $\text{DFS}(G)$  gilt für  $\{u, v\} \in \binom{V}{2}$  genau eine der Bedingungen

(i)

(ii)

(iii)



# Tiefensuche – Analyse

d.h. für jedes Paar  $\{u, v\}$  von Knoten (mit  $u \neq v$ )

**Satz.**

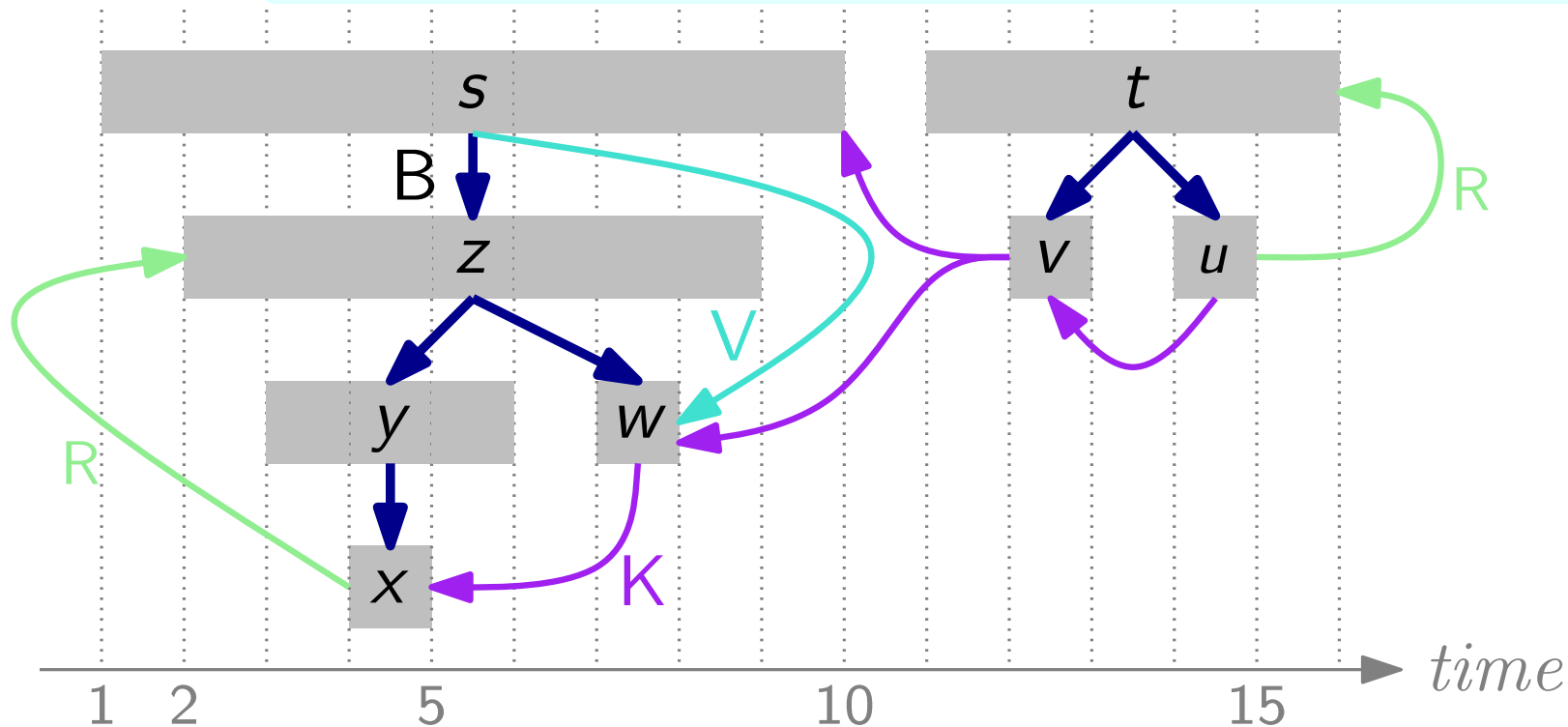
(Klammertheorem)

Nach  $\text{DFS}(G)$  gilt für  $\{u, v\} \in \binom{V}{2}$  genau eine der Bedingungen

(i)

(ii)

(iii)



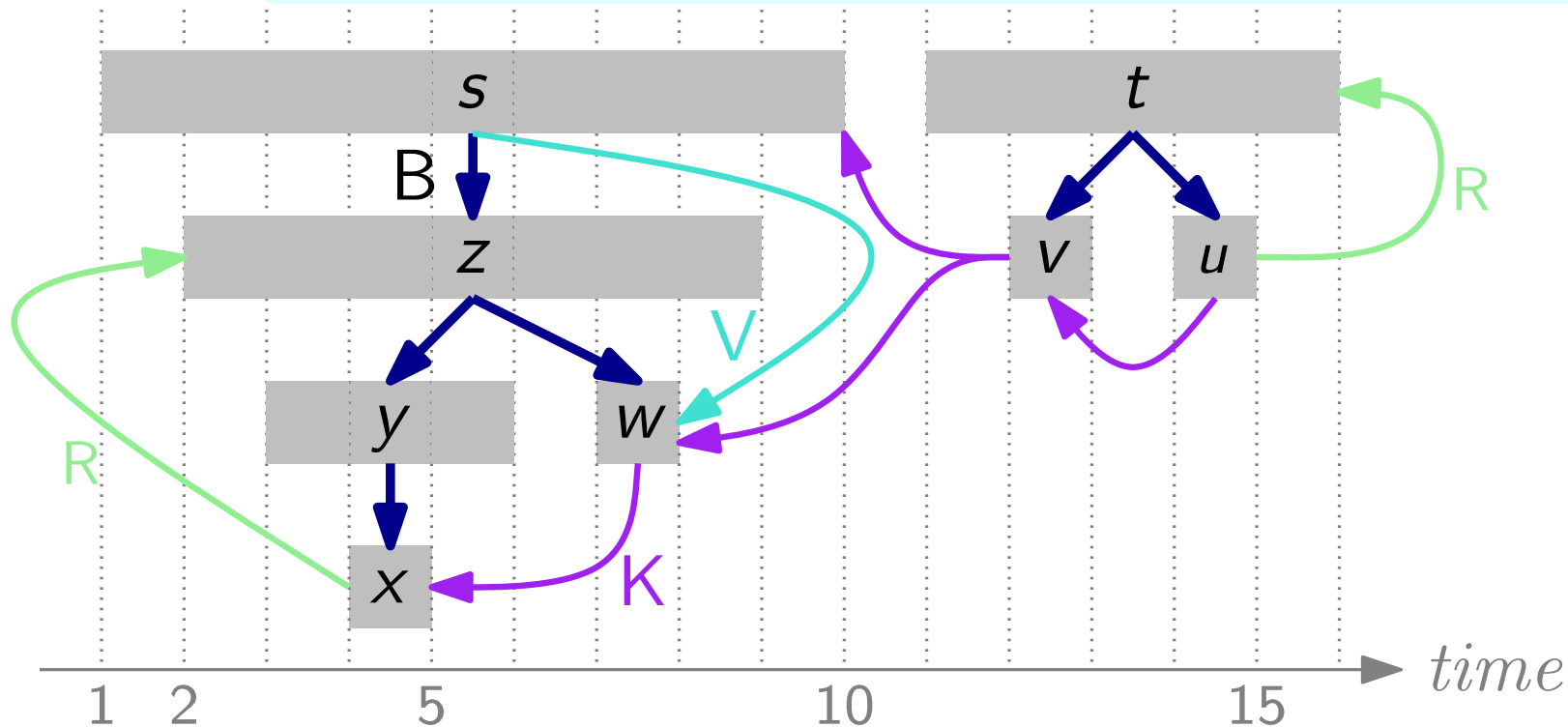
# Tiefensuche – Analyse

## Satz.

(Klammertheorem)

Nach  $\text{DFS}(G)$  gilt für  $\{u, v\} \in \binom{V}{2}$  genau eine der Bedingungen

- (i) Besuchsintervalle disjunkt und Baumkanten enthalten weder  $u$ - $v$ - noch  $v$ - $u$ -Weg.
- (ii)
- (iii)



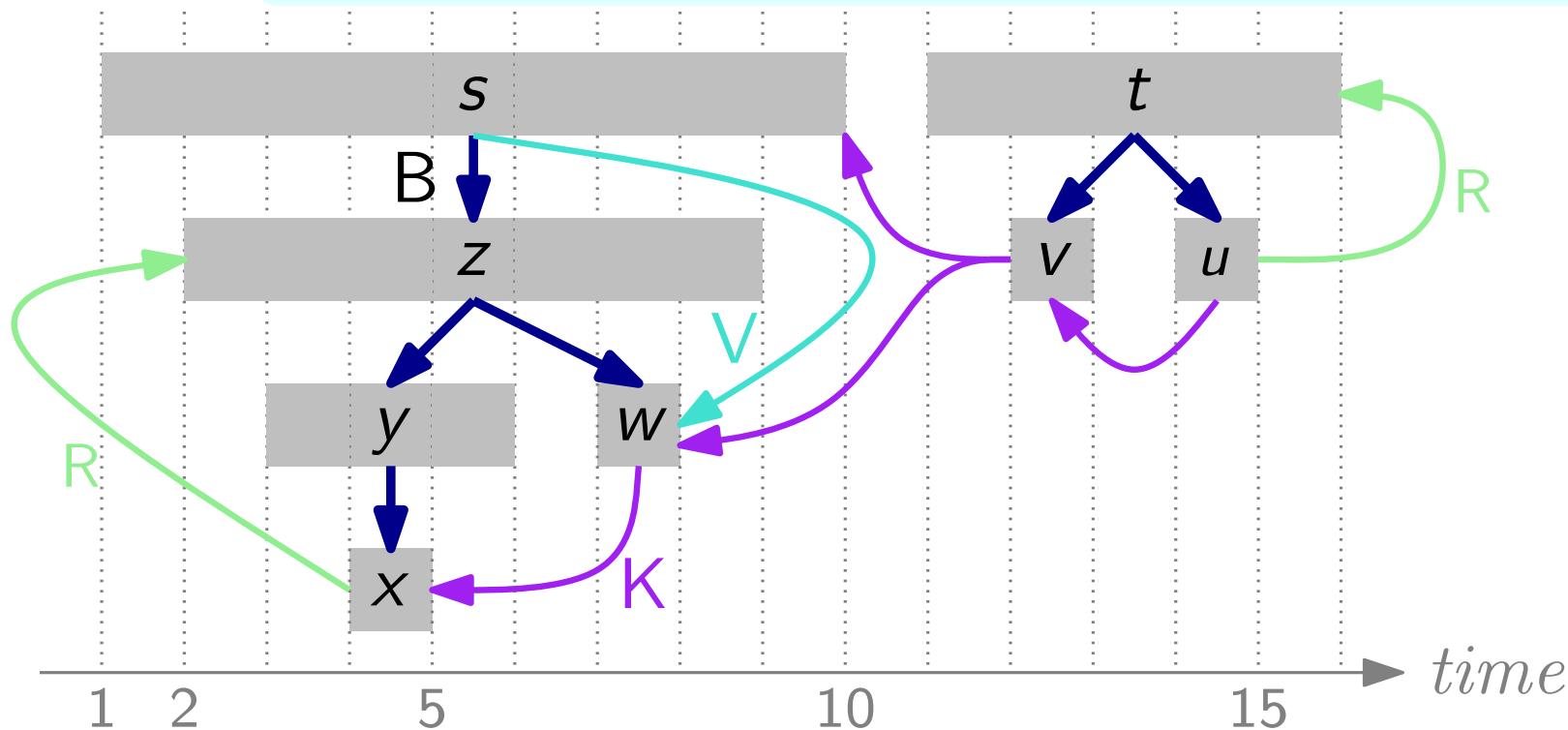
# Tiefensuche – Analyse

## Satz.

(Klammerentheorem)

Nach  $\text{DFS}(G)$  gilt für  $\{u, v\} \in \binom{V}{2}$  genau eine der Bedingungen

- (i) Besuchsintervalle disjunkt und Baumkanten enthalten weder  $u$ - $v$ - noch  $v$ - $u$ -Weg.
- (ii)  $[u.d, u.f] \subset [v.d, v.f]$  und Baumkanten enthalten  $v$ - $u$ -Weg.
- (iii)



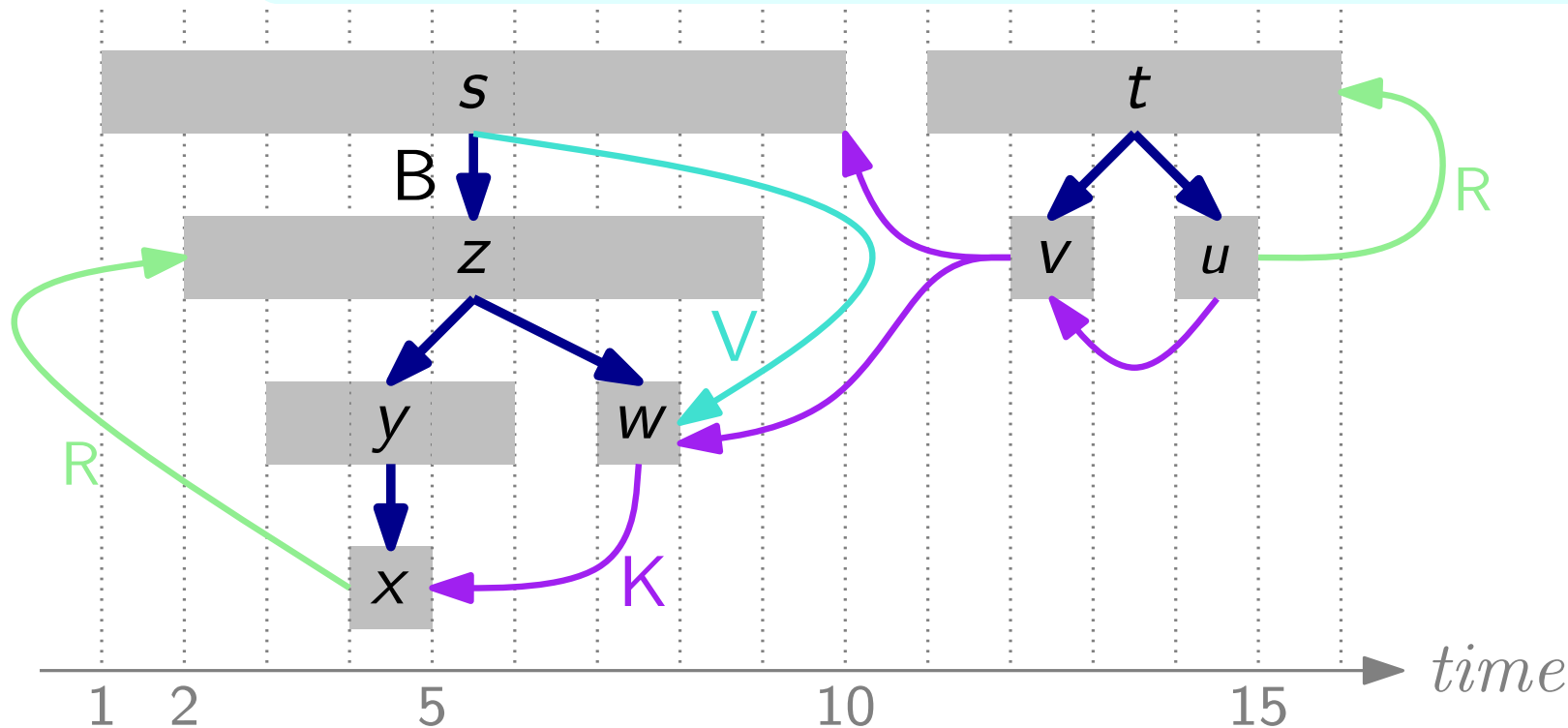
# Tiefensuche – Analyse

## Satz.

(Klammerentheorem)

Nach  $\text{DFS}(G)$  gilt für  $\{u, v\} \in \binom{V}{2}$  genau eine der Bedingungen

- (i) Besuchsintervalle disjunkt und Baumkanten enthalten weder  $u$ - $v$ - noch  $v$ - $u$ -Weg.
- (ii)  $[u.d, u.f] \subset [v.d, v.f]$  und Baumkanten enthalten  $v$ - $u$ -Weg.
- (iii) Wie (ii), nur umgekehrt.



# Tiefensuche – Analyse

**Satz.** (Klammertheorem)

Nach  $\text{DFS}(G)$  gilt für  $\{u, v\} \in \binom{V}{2}$  genau eine der Bedingungen

- (i) Besuchsintervalle disjunkt und Baumkanten enthalten weder  $u$ - $v$ - noch  $v$ - $u$ -Weg.
- (ii)  $[u.d, u.f] \subset [v.d, v.f]$  und Baumkanten enthalten  $v$ - $u$ -Weg.
- (iii) Wie (ii), nur umgekehrt.

*Beweis.* Wir betrachten zwei Fälle.

```
DFSVisit(Graph G, Vertex u)
time = time + 1
u.d = time; u.color = gray
foreach v ∈ Adj[u] do
  if v.color == white then
    v.π = u; DFSVisit(G, v)
time = time + 1
u.f = time; u.color = black
```

# Tiefensuche – Analyse

```
DFSVisit(Graph G, Vertex u)
  time = time + 1
  u.d = time; u.color = gray
  foreach v ∈ Adj[u] do
    if v.color == white then
      v.π = u; DFSVisit(G, v)
  time = time + 1
  u.f = time; u.color = black
```

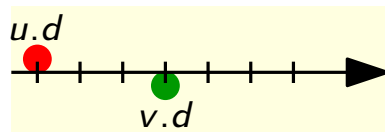
**Satz.** (Klammertheorem)

Nach  $\text{DFS}(G)$  gilt für  $\{u, v\} \in \binom{V}{2}$  genau eine der Bedingungen

- (i) Besuchsintervalle disjunkt und Baumkanten enthalten weder  $u$ - $v$ - noch  $v$ - $u$ -Weg.
- (ii)  $[u.d, u.f] \subset [v.d, v.f]$  und Baumkanten enthalten  $v$ - $u$ -Weg.
- (iii) Wie (ii), nur umgekehrt.

**Beweis.** Wir betrachten zwei Fälle.

1. Fall:  $u.d < v.d$ .



# Tiefensuche – Analyse

```
DFSVisit(Graph G, Vertex u)
time = time + 1
u.d = time; u.color = gray
foreach v ∈ Adj[u] do
  if v.color == white then
    v.π = u; DFSVisit(G, v)
time = time + 1
u.f = time; u.color = black
```

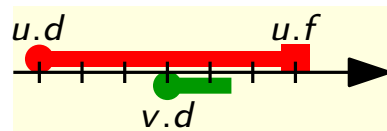
**Satz.** (Klammertheorem)

Nach  $\text{DFS}(G)$  gilt für  $\{u, v\} \in \binom{V}{2}$  genau eine der Bedingungen

- (i) Besuchsintervalle disjunkt und Baumkanten enthalten weder  $u$ - $v$ - noch  $v$ - $u$ -Weg.
- (ii)  $[u.d, u.f] \subset [v.d, v.f]$  und Baumkanten enthalten  $v$ - $u$ -Weg.
- (iii) Wie (ii), nur umgekehrt.

**Beweis.** Wir betrachten zwei Fälle.

1. Fall:  $u.d < v.d$ .



A)  $v.d < u.f$ .



# Tiefensuche – Analyse

```
DFSVisit(Graph G, Vertex u)
time = time + 1
u.d = time; u.color = gray
foreach v ∈ Adj[u] do
  if v.color == white then
    v.π = u; DFSVisit(G, v)
time = time + 1
u.f = time; u.color = black
```

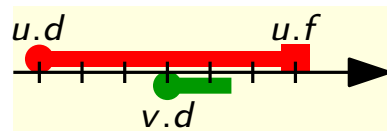
**Satz.** (Klammertheorem)

Nach  $\text{DFS}(G)$  gilt für  $\{u, v\} \in \binom{V}{2}$  genau eine der Bedingungen

- (i) Besuchsintervalle disjunkt und Baumkanten enthalten weder  $u$ - $v$ - noch  $v$ - $u$ -Weg.
- (ii)  $[u.d, u.f] \subset [v.d, v.f]$  und Baumkanten enthalten  $v$ - $u$ -Weg.
- (iii) Wie (ii), nur umgekehrt.

**Beweis.** Wir betrachten zwei Fälle.

1. Fall:  $u.d < v.d$ .



A)  $v.d < u.f$ , d.h.  $v$  wurde entdeckt, als  $u$  noch grau war.

# Tiefensuche – Analyse

```
DFSVisit(Graph G, Vertex u)
time = time + 1
u.d = time; u.color = gray
foreach v ∈ Adj[u] do
  if v.color == white then
    v.π = u; DFSVisit(G, v)
time = time + 1
u.f = time; u.color = black
```

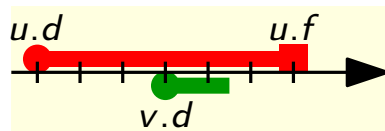
**Satz.** (Klammertheorem)

Nach  $\text{DFS}(G)$  gilt für  $\{u, v\} \in \binom{V}{2}$  genau eine der Bedingungen

- (i) Besuchsintervalle disjunkt und Baumkanten enthalten weder  $u$ - $v$ - noch  $v$ - $u$ -Weg.
- (ii)  $[u.d, u.f] \subset [v.d, v.f]$  und Baumkanten enthalten  $v$ - $u$ -Weg.
- (iii) Wie (ii), nur umgekehrt.

**Beweis.** Wir betrachten zwei Fälle.

1. Fall:  $u.d < v.d$ .



A)  $v.d < u.f$ , d.h.  $v$  wurde entdeckt, als  $u$  noch grau war.

$\Rightarrow v$  ist *Nachfolger* von  $u$

# Tiefensuche – Analyse

```
DFSVisit(Graph G, Vertex u)
time = time + 1
u.d = time; u.color = gray
foreach v ∈ Adj[u] do
  if v.color == white then
    v.π = u; DFSVisit(G, v)
time = time + 1
u.f = time; u.color = black
```

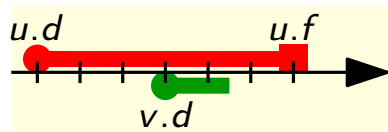
**Satz.** (Klammertheorem)

Nach  $\text{DFS}(G)$  gilt für  $\{u, v\} \in \binom{V}{2}$  genau eine der Bedingungen

- (i) Besuchsintervalle disjunkt und Baumkanten enthalten weder  $u$ - $v$ - noch  $v$ - $u$ -Weg.
- (ii)  $[u.d, u.f] \subset [v.d, v.f]$  und Baumkanten enthalten  $v$ - $u$ -Weg.
- (iii) Wie (ii), nur umgekehrt.

**Beweis.** Wir betrachten zwei Fälle.

1. Fall:  $u.d < v.d$ .



- A)  $v.d < u.f$ , d.h.  $v$  wurde entdeckt, als  $u$  noch grau war.  
 $\Rightarrow v$  ist *Nachfolger* von  $u$ , d.h. es gibt einen  $u$ - $v$ -Weg.

# Tiefensuche – Analyse

```
DFSVisit(Graph G, Vertex u)
time = time + 1
u.d = time; u.color = gray
foreach v ∈ Adj[u] do
  if v.color == white then
    v.π = u; DFSVisit(G, v)
time = time + 1
u.f = time; u.color = black
```

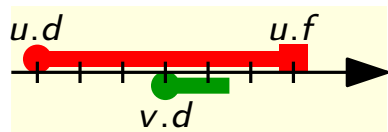
**Satz.** (Klammertheorem)

Nach  $\text{DFS}(G)$  gilt für  $\{u, v\} \in \binom{V}{2}$  genau eine der Bedingungen

- (i) Besuchsintervalle disjunkt und Baumkanten enthalten weder  $u$ - $v$ - noch  $v$ - $u$ -Weg.
- (ii)  $[u.d, u.f] \subset [v.d, v.f]$  und Baumkanten enthalten  $v$ - $u$ -Weg.
- (iii) Wie (ii), nur umgekehrt.

**Beweis.** Wir betrachten zwei Fälle.

1. Fall:  $u.d < v.d$ .



A)  $v.d < u.f$ , d.h.  $v$  wurde entdeckt, als  $u$  noch grau war.  
 $\Rightarrow v$  ist *Nachfolger* von  $u$ , d.h. es gibt einen  $u$ - $v$ -Weg.

Wegen  $u.d < v.d$  gilt:

# Tiefensuche – Analyse

```
DFSVisit(Graph G, Vertex u)
  time = time + 1
  u.d = time; u.color = gray
  foreach v ∈ Adj[u] do
    if v.color == white then
      v.π = u; DFSVisit(G, v)
  time = time + 1
  u.f = time; u.color = black
```

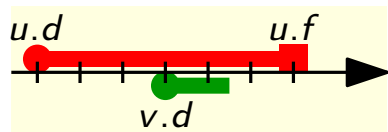
**Satz.** (Klammertheorem)

Nach  $\text{DFS}(G)$  gilt für  $\{u, v\} \in \binom{V}{2}$  genau eine der Bedingungen

- (i) Besuchsintervalle disjunkt und Baumkanten enthalten weder  $u$ - $v$ - noch  $v$ - $u$ -Weg.
- (ii)  $[u.d, u.f] \subset [v.d, v.f]$  und Baumkanten enthalten  $v$ - $u$ -Weg.
- (iii) Wie (ii), nur umgekehrt.

**Beweis.** Wir betrachten zwei Fälle.

1. Fall:  $u.d < v.d$ .



A)  $v.d < u.f$ , d.h.  $v$  wurde entdeckt, als  $u$  noch grau war.  
 $\Rightarrow v$  ist *Nachfolger* von  $u$ , d.h. es gibt einen  $u$ - $v$ -Weg.

Wegen  $u.d < v.d$  gilt:  $v$  wurde später als  $u$  entdeckt.

# Tiefensuche – Analyse

```
DFSVisit(Graph G, Vertex u)
  time = time + 1
  u.d = time; u.color = gray
  foreach v ∈ Adj[u] do
    if v.color == white then
      v.π = u; DFSVisit(G, v)
  time = time + 1
  u.f = time; u.color = black
```

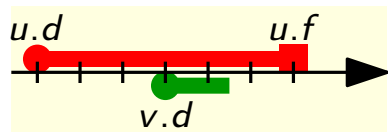
**Satz.** (Klammertheorem)

Nach  $\text{DFS}(G)$  gilt für  $\{u, v\} \in \binom{V}{2}$  genau eine der Bedingungen

- (i) Besuchsintervalle disjunkt und Baumkanten enthalten weder  $u$ - $v$ - noch  $v$ - $u$ -Weg.
- (ii)  $[u.d, u.f] \subset [v.d, v.f]$  und Baumkanten enthalten  $v$ - $u$ -Weg.
- (iii) Wie (ii), nur umgekehrt.

**Beweis.** Wir betrachten zwei Fälle.

1. Fall:  $u.d < v.d$ .



A)  $v.d < u.f$ , d.h.  $v$  wurde entdeckt, als  $u$  noch grau war.  
 $\Rightarrow v$  ist *Nachfolger* von  $u$ , d.h. es gibt einen  $u$ - $v$ -Weg.

Wegen  $u.d < v.d$  gilt:  $v$  wurde später als  $u$  entdeckt.

$\Rightarrow$  alle Kanten, die  $v$  verlassen, sind erforscht;

$v$  wird schwarz, *bevor* DFS zu  $u$  zurückkehrt und  $u$  schwarz macht

# Tiefensuche – Analyse

```
DFSVisit(Graph G, Vertex u)
  time = time + 1
  u.d = time; u.color = gray
  foreach v ∈ Adj[u] do
    if v.color == white then
      v.π = u; DFSVisit(G, v)
  time = time + 1
  u.f = time; u.color = black
```

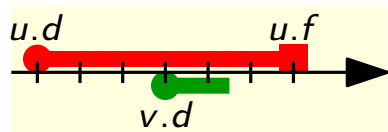
**Satz.** (Klammertheorem)

Nach  $\text{DFS}(G)$  gilt für  $\{u, v\} \in \binom{V}{2}$  genau eine der Bedingungen

- (i) Besuchsintervalle disjunkt und Baumkanten enthalten weder  $u$ - $v$ - noch  $v$ - $u$ -Weg.
- (ii)  $[u.d, u.f] \subset [v.d, v.f]$  und Baumkanten enthalten  $v$ - $u$ -Weg.
- (iii) Wie (ii), nur umgekehrt.

**Beweis.** Wir betrachten zwei Fälle.

1. Fall:  $u.d < v.d$ .



A)  $v.d < u.f$ , d.h.  $v$  wurde entdeckt, als  $u$  noch grau war.  
 $\Rightarrow v$  ist *Nachfolger* von  $u$ , d.h. es gibt einen  $u$ - $v$ -Weg.

Wegen  $u.d < v.d$  gilt:  $v$  wurde später als  $u$  entdeckt.

$\Rightarrow$  alle Kanten, die  $v$  verlassen, sind erforscht;

$v$  wird schwarz, *bevor* DFS zu  $u$  zurückkehrt und  $u$  schwarz macht  $\Rightarrow [v.d, v.f] \subset [u.d, u.f]$ ,

# Tiefensuche – Analyse

```
DFSVisit(Graph G, Vertex u)
  time = time + 1
  u.d = time; u.color = gray
  foreach v ∈ Adj[u] do
    if v.color == white then
      v.π = u; DFSVisit(G, v)
  time = time + 1
  u.f = time; u.color = black
```

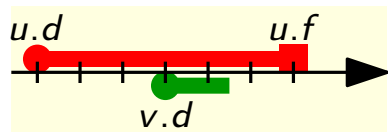
**Satz.** (Klammertheorem)

Nach  $\text{DFS}(G)$  gilt für  $\{u, v\} \in \binom{V}{2}$  genau eine der Bedingungen

- (i) Besuchsintervalle disjunkt und Baumkanten enthalten weder  $u$ - $v$ - noch  $v$ - $u$ -Weg.
- (ii)  $[u.d, u.f] \subset [v.d, v.f]$  und Baumkanten enthalten  $v$ - $u$ -Weg.
- (iii) Wie (ii), nur umgekehrt.

**Beweis.** Wir betrachten zwei Fälle.

1. Fall:  $u.d < v.d$ .



A)  $v.d < u.f$ , d.h.  $v$  wurde entdeckt, als  $u$  noch grau war.  
 $\Rightarrow v$  ist *Nachfolger* von  $u$ , d.h. es gibt einen  $u$ - $v$ -Weg.

Wegen  $u.d < v.d$  gilt:  $v$  wurde später als  $u$  entdeckt.

$\Rightarrow$  alle Kanten, die  $v$  verlassen, sind erforscht;

$v$  wird schwarz, *bevor* DFS zu  $u$  zurückkehrt und  $u$

schwarz macht  $\Rightarrow [v.d, v.f] \subset [u.d, u.f]$ , d.h. (iii) ✓



# Tiefensuche – Analyse

```
DFSVisit(Graph G, Vertex u)
  time = time + 1
  u.d = time; u.color = gray
  foreach v ∈ Adj[u] do
    if v.color == white then
      v.π = u; DFSVisit(G, v)
  time = time + 1
  u.f = time; u.color = black
```

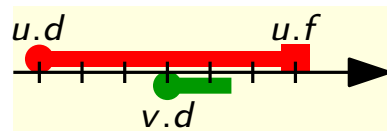
**Satz.** (Klammertheorem)

Nach  $\text{DFS}(G)$  gilt für  $\{u, v\} \in \binom{V}{2}$  genau eine der Bedingungen

- (i) Besuchsintervalle disjunkt und Baumkanten enthalten weder  $u$ - $v$ - noch  $v$ - $u$ -Weg.
- (ii)  $[u.d, u.f] \subset [v.d, v.f]$  und Baumkanten enthalten  $v$ - $u$ -Weg.
- (iii) Wie (ii), nur umgekehrt.

**Beweis.** Wir betrachten zwei Fälle.

1. Fall:  $u.d < v.d$ .



A)  $v.d < u.f$ . ✓

# Tiefensuche – Analyse

```
DFSVisit(Graph G, Vertex u)
time = time + 1
u.d = time; u.color = gray
foreach v ∈ Adj[u] do
  if v.color == white then
    v.π = u; DFSVisit(G, v)
time = time + 1
u.f = time; u.color = black
```

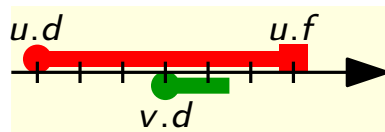
**Satz.** (Klammertheorem)

Nach  $\text{DFS}(G)$  gilt für  $\{u, v\} \in \binom{V}{2}$  genau eine der Bedingungen

- (i) Besuchsintervalle disjunkt und Baumkanten enthalten weder  $u$ - $v$ - noch  $v$ - $u$ -Weg.
- (ii)  $[u.d, u.f] \subset [v.d, v.f]$  und Baumkanten enthalten  $v$ - $u$ -Weg.
- (iii) Wie (ii), nur umgekehrt.

**Beweis.** Wir betrachten zwei Fälle.

1. Fall:  $u.d < v.d$ .



A)  $v.d < u.f$ . ✓

B)

# Tiefensuche – Analyse

```
DFSVisit(Graph G, Vertex u)
time = time + 1
u.d = time; u.color = gray
foreach v ∈ Adj[u] do
  if v.color == white then
    v.π = u; DFSVisit(G, v)
time = time + 1
u.f = time; u.color = black
```

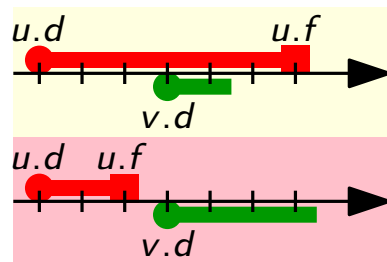
**Satz.** (Klammertheorem)

Nach  $\text{DFS}(G)$  gilt für  $\{u, v\} \in \binom{V}{2}$  genau eine der Bedingungen

- (i) Besuchsintervalle disjunkt und Baumkanten enthalten weder  $u$ - $v$ - noch  $v$ - $u$ -Weg.
- (ii)  $[u.d, u.f] \subset [v.d, v.f]$  und Baumkanten enthalten  $v$ - $u$ -Weg.
- (iii) Wie (ii), nur umgekehrt.

**Beweis.** Wir betrachten zwei Fälle.

1. Fall:  $u.d < v.d$ .



A)  $v.d < u.f$ . ✓

B)  $u.f < v.d$ .

# Tiefensuche – Analyse

```
DFSVisit(Graph G, Vertex u)
time = time + 1
u.d = time; u.color = gray
foreach v ∈ Adj[u] do
  if v.color == white then
    v.π = u; DFSVisit(G, v)
time = time + 1
u.f = time; u.color = black
```

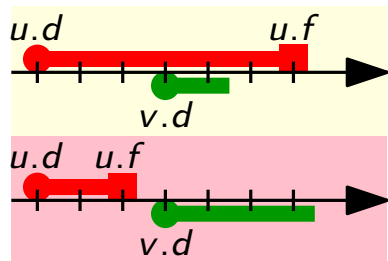
**Satz.** (Klammertheorem)

Nach  $\text{DFS}(G)$  gilt für  $\{u, v\} \in \binom{V}{2}$  genau eine der Bedingungen

- (i) Besuchsintervalle disjunkt und Baumkanten enthalten weder  $u$ - $v$ - noch  $v$ - $u$ -Weg.
- (ii)  $[u.d, u.f] \subset [v.d, v.f]$  und Baumkanten enthalten  $v$ - $u$ -Weg.
- (iii) Wie (ii), nur umgekehrt.

**Beweis.** Wir betrachten zwei Fälle.

1. Fall:  $u.d < v.d$ .



A)  $v.d < u.f$ . ✓

B)  $u.f < v.d$ .

$u.f < v.d$

# Tiefensuche – Analyse

```
DFSVisit(Graph G, Vertex u)
  time = time + 1
  u.d = time; u.color = gray
  foreach v ∈ Adj[u] do
    if v.color == white then
      v.π = u; DFSVisit(G, v)
  time = time + 1
  u.f = time; u.color = black
```

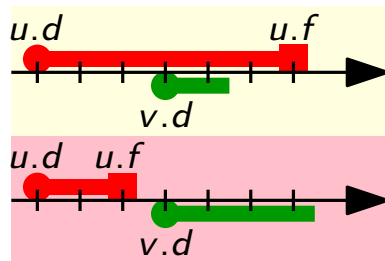
**Satz.** (Klammertheorem)

Nach  $\text{DFS}(G)$  gilt für  $\{u, v\} \in \binom{V}{2}$  genau eine der Bedingungen

- (i) Besuchsintervalle disjunkt und Baumkanten enthalten weder  $u$ - $v$ - noch  $v$ - $u$ -Weg.
- (ii)  $[u.d, u.f] \subset [v.d, v.f]$  und Baumkanten enthalten  $v$ - $u$ -Weg.
- (iii) Wie (ii), nur umgekehrt.

**Beweis.** Wir betrachten zwei Fälle.

1. Fall:  $u.d < v.d$ .



A)  $v.d < u.f$ . ✓

B)  $u.f < v.d$ .

Laut Code gilt außerdem

$u.f < v.d$

# Tiefensuche – Analyse

```
DFSVisit(Graph G, Vertex u)
  time = time + 1
  u.d = time; u.color = gray
  foreach v ∈ Adj[u] do
    if v.color == white then
      v.π = u; DFSVisit(G, v)
  time = time + 1
  u.f = time; u.color = black
```

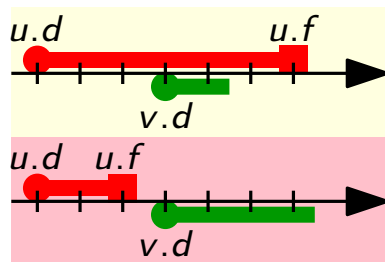
**Satz.** (Klammertheorem)

Nach  $\text{DFS}(G)$  gilt für  $\{u, v\} \in \binom{V}{2}$  genau eine der Bedingungen

- (i) Besuchsintervalle disjunkt und Baumkanten enthalten weder  $u$ - $v$ - noch  $v$ - $u$ -Weg.
- (ii)  $[u.d, u.f] \subset [v.d, v.f]$  und Baumkanten enthalten  $v$ - $u$ -Weg.
- (iii) Wie (ii), nur umgekehrt.

**Beweis.** Wir betrachten zwei Fälle.

1. Fall:  $u.d < v.d$ .



A)  $v.d < u.f$ . ✓

B)  $u.f < v.d$ .

Laut Code gilt außerdem  $u.d < u.f < v.d$

# Tiefensuche – Analyse

```
DFSVisit(Graph G, Vertex u)
  time = time + 1
  u.d = time; u.color = gray
  foreach v ∈ Adj[u] do
    if v.color == white then
      v.π = u; DFSVisit(G, v)
  time = time + 1
  u.f = time; u.color = black
```

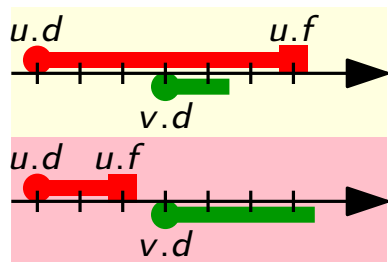
**Satz.** (Klammertheorem)

Nach  $\text{DFS}(G)$  gilt für  $\{u, v\} \in \binom{V}{2}$  genau eine der Bedingungen

- (i) Besuchsintervalle disjunkt und Baumkanten enthalten weder  $u$ - $v$ - noch  $v$ - $u$ -Weg.
- (ii)  $[u.d, u.f] \subset [v.d, v.f]$  und Baumkanten enthalten  $v$ - $u$ -Weg.
- (iii) Wie (ii), nur umgekehrt.

**Beweis.** Wir betrachten zwei Fälle.

1. Fall:  $u.d < v.d$ .



A)  $v.d < u.f$ . ✓

B)  $u.f < v.d$ .

Laut Code gilt außerdem  $u.d < u.f < v.d < v.f$

# Tiefensuche – Analyse

```
DFSVisit(Graph G, Vertex u)
  time = time + 1
  u.d = time; u.color = gray
  foreach v ∈ Adj[u] do
    if v.color == white then
      v.π = u; DFSVisit(G, v)
  time = time + 1
  u.f = time; u.color = black
```

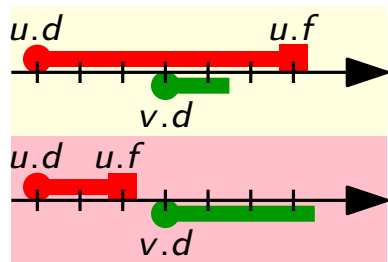
**Satz.** (Klammertheorem)

Nach  $\text{DFS}(G)$  gilt für  $\{u, v\} \in \binom{V}{2}$  genau eine der Bedingungen

- (i) Besuchsintervalle disjunkt und Baumkanten enthalten weder  $u$ - $v$ - noch  $v$ - $u$ -Weg.
- (ii)  $[u.d, u.f] \subset [v.d, v.f]$  und Baumkanten enthalten  $v$ - $u$ -Weg.
- (iii) Wie (ii), nur umgekehrt.

**Beweis.** Wir betrachten zwei Fälle.

1. Fall:  $u.d < v.d$ .



A)  $v.d < u.f$ . ✓

B)  $u.f < v.d$ .

Laut Code gilt außerdem  $u.d < u.f < v.d < v.f$

⇒



# Tiefensuche – Analyse

```
DFSVisit(Graph G, Vertex u)
  time = time + 1
  u.d = time; u.color = gray
  foreach v ∈ Adj[u] do
    if v.color == white then
      v.π = u; DFSVisit(G, v)
  time = time + 1
  u.f = time; u.color = black
```

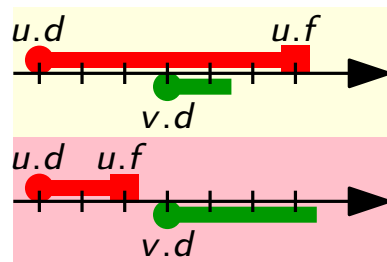
**Satz.** (Klammertheorem)

Nach  $\text{DFS}(G)$  gilt für  $\{u, v\} \in \binom{V}{2}$  genau eine der Bedingungen

- (i) Besuchsintervalle disjunkt und Baumkanten enthalten weder  $u$ - $v$ - noch  $v$ - $u$ -Weg.
- (ii)  $[u.d, u.f] \subset [v.d, v.f]$  und Baumkanten enthalten  $v$ - $u$ -Weg.
- (iii) Wie (ii), nur umgekehrt.

**Beweis.** Wir betrachten zwei Fälle.

1. Fall:  $u.d < v.d$ .



A)  $v.d < u.f$ . ✓

B)  $u.f < v.d$ .

Laut Code gilt außerdem  $u.d < u.f < v.d < v.f$

$\Rightarrow [u.d, u.f] \cap [v.d, v.f] = \emptyset$

# Tiefensuche – Analyse

```
DFSVisit(Graph G, Vertex u)
  time = time + 1
  u.d = time; u.color = gray
  foreach v ∈ Adj[u] do
    if v.color == white then
      v.π = u; DFSVisit(G, v)
  time = time + 1
  u.f = time; u.color = black
```

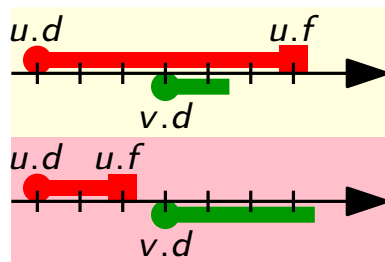
**Satz.** (Klammertheorem)

Nach  $\text{DFS}(G)$  gilt für  $\{u, v\} \in \binom{V}{2}$  genau eine der Bedingungen

- (i) Besuchsintervalle disjunkt und Baumkanten enthalten weder  $u$ - $v$ - noch  $v$ - $u$ -Weg.
- (ii)  $[u.d, u.f] \subset [v.d, v.f]$  und Baumkanten enthalten  $v$ - $u$ -Weg.
- (iii) Wie (ii), nur umgekehrt.

**Beweis.** Wir betrachten zwei Fälle.

1. Fall:  $u.d < v.d$ .



A)  $v.d < u.f$ . ✓

B)  $u.f < v.d$ .

Laut Code gilt außerdem  $u.d < u.f < v.d < v.f$

$$\Rightarrow [u.d, u.f] \cap [v.d, v.f] = \emptyset$$

$\Rightarrow$  Keiner der beiden Knoten wurde entdeckt, während der andere noch grau war.

# Tiefensuche – Analyse

```
DFSVisit(Graph G, Vertex u)
  time = time + 1
  u.d = time; u.color = gray
  foreach v ∈ Adj[u] do
    if v.color == white then
      v.π = u; DFSVisit(G, v)
  time = time + 1
  u.f = time; u.color = black
```

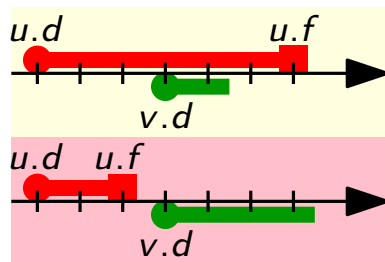
**Satz.** (Klammertheorem)

Nach  $\text{DFS}(G)$  gilt für  $\{u, v\} \in \binom{V}{2}$  genau eine der Bedingungen

- (i) Besuchsintervalle disjunkt und Baumkanten enthalten weder  $u$ - $v$ - noch  $v$ - $u$ -Weg.
- (ii)  $[u.d, u.f] \subset [v.d, v.f]$  und Baumkanten enthalten  $v$ - $u$ -Weg.
- (iii) Wie (ii), nur umgekehrt.

**Beweis.** Wir betrachten zwei Fälle.

1. Fall:  $u.d < v.d$ .



A)  $v.d < u.f$ . ✓

B)  $u.f < v.d$ .

Laut Code gilt außerdem  $u.d < u.f < v.d < v.f$

$$\Rightarrow [u.d, u.f] \cap [v.d, v.f] = \emptyset$$

$\Rightarrow$  Keiner der beiden Knoten wurde entdeckt, während der andere noch grau war, d.h. keiner Nachf. des anderen.

# Tiefensuche – Analyse

```
DFSVisit(Graph G, Vertex u)
  time = time + 1
  u.d = time; u.color = gray
  foreach v ∈ Adj[u] do
    if v.color == white then
      v.π = u; DFSVisit(G, v)
  time = time + 1
  u.f = time; u.color = black
```

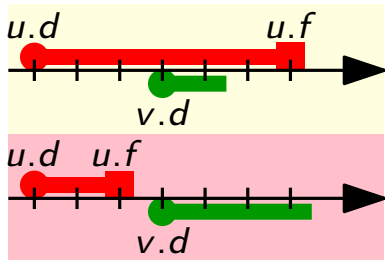
**Satz.** (Klammertheorem)

Nach  $\text{DFS}(G)$  gilt für  $\{u, v\} \in \binom{V}{2}$  genau eine der Bedingungen

- (i) Besuchsintervalle disjunkt und Baumkanten enthalten weder  $u$ - $v$ - noch  $v$ - $u$ -Weg.
- (ii)  $[u.d, u.f] \subset [v.d, v.f]$  und Baumkanten enthalten  $v$ - $u$ -Weg.
- (iii) Wie (ii), nur umgekehrt.

**Beweis.** Wir betrachten zwei Fälle.

1. Fall:  $u.d < v.d$ .



A)  $v.d < u.f$ . ✓

B)  $u.f < v.d$ .

Laut Code gilt außerdem  $u.d < u.f < v.d < v.f$

$$\Rightarrow [u.d, u.f] \cap [v.d, v.f] = \emptyset$$

$\Rightarrow$  Keiner der beiden Knoten wurde entdeckt, während der andere noch grau war, d.h. keiner Nachf. des anderen. (i)  $\uparrow\uparrow$

# Tiefensuche – Analyse

```
DFSVisit(Graph G, Vertex u)
  time = time + 1
  u.d = time; u.color = gray
  foreach v ∈ Adj[u] do
    if v.color == white then
      v.π = u; DFSVisit(G, v)
  time = time + 1
  u.f = time; u.color = black
```

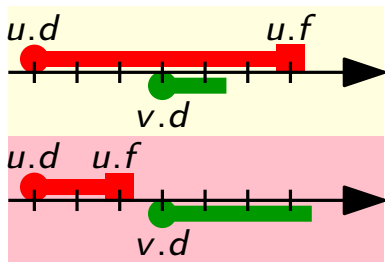
**Satz.** (Klammertheorem)

Nach  $\text{DFS}(G)$  gilt für  $\{u, v\} \in \binom{V}{2}$  genau eine der Bedingungen

- (i) Besuchsintervalle disjunkt und Baumkanten enthalten weder  $u$ - $v$ - noch  $v$ - $u$ -Weg.
- (ii)  $[u.d, u.f] \subset [v.d, v.f]$  und Baumkanten enthalten  $v$ - $u$ -Weg.
- (iii) Wie (ii), nur umgekehrt.

**Beweis.** Wir betrachten zwei Fälle.

1. Fall:  $u.d < v.d$ .



A)  $v.d < u.f$ . ✓

B)  $u.f < v.d$ . ✓

Laut Code gilt außerdem  $u.d < u.f < v.d < v.f$

$$\Rightarrow [u.d, u.f] \cap [v.d, v.f] = \emptyset$$

$\Rightarrow$  Keiner der beiden Knoten wurde entdeckt, während der andere noch grau war, d.h. keiner Nachf. des anderen. (i)  $\uparrow\uparrow$

# Tiefensuche – Analyse

```
DFSVisit(Graph G, Vertex u)
  time = time + 1
  u.d = time; u.color = gray
  foreach v ∈ Adj[u] do
    if v.color == white then
      v.π = u; DFSVisit(G, v)
  time = time + 1
  u.f = time; u.color = black
```

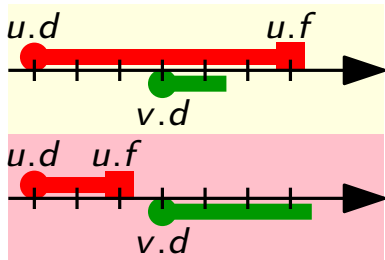
**Satz.** (Klammertheorem)

Nach  $\text{DFS}(G)$  gilt für  $\{u, v\} \in \binom{V}{2}$  genau eine der Bedingungen

- (i) Besuchsintervalle disjunkt und Baumkanten enthalten weder  $u$ - $v$ - noch  $v$ - $u$ -Weg.
- (ii)  $[u.d, u.f] \subset [v.d, v.f]$  und Baumkanten enthalten  $v$ - $u$ -Weg.
- (iii) Wie (ii), nur umgekehrt.

**Beweis.** Wir betrachten zwei Fälle.

1. Fall:  $u.d < v.d$ . ✓



A)  $v.d < u.f$ . ✓

B)  $u.f < v.d$ . ✓

Laut Code gilt außerdem  $u.d < u.f < v.d < v.f$

$$\Rightarrow [u.d, u.f] \cap [v.d, v.f] = \emptyset$$

$\Rightarrow$  Keiner der beiden Knoten wurde entdeckt, während der andere noch grau war, d.h. keiner Nachf. des anderen. (i) ↑↑

# Tiefensuche – Analyse

```
DFSVisit(Graph G, Vertex u)
  time = time + 1
  u.d = time; u.color = gray
  foreach v ∈ Adj[u] do
    if v.color == white then
      v.π = u; DFSVisit(G, v)
  time = time + 1
  u.f = time; u.color = black
```

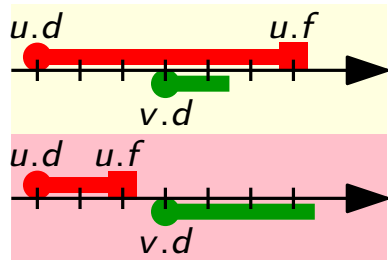
**Satz.** (Klammertheorem)

Nach  $\text{DFS}(G)$  gilt für  $\{u, v\} \in \binom{V}{2}$  genau eine der Bedingungen

- (i) Besuchsintervalle disjunkt und Baumkanten enthalten weder  $u$ - $v$ - noch  $v$ - $u$ -Weg.
- (ii)  $[u.d, u.f] \subset [v.d, v.f]$  und Baumkanten enthalten  $v$ - $u$ -Weg.
- (iii) Wie (ii), nur umgekehrt.

**Beweis.** Wir betrachten zwei Fälle.

1. Fall:  $u.d < v.d$ . ✓ 2. Fall:  $v.d < u.d$ .



A)  $v.d < u.f$ . ✓

B)  $u.f < v.d$ . ✓

Laut Code gilt außerdem  $u.d < u.f < v.d < v.f$

$$\Rightarrow [u.d, u.f] \cap [v.d, v.f] = \emptyset$$

$\Rightarrow$  Keiner der beiden Knoten wurde entdeckt, während der andere noch grau war, d.h. keiner Nachf. des anderen. (i)  $\uparrow\uparrow$

# Tiefensuche – Analyse

```
DFSVisit(Graph G, Vertex u)
  time = time + 1
  u.d = time; u.color = gray
  foreach v ∈ Adj[u] do
    if v.color == white then
      v.π = u; DFSVisit(G, v)
  time = time + 1
  u.f = time; u.color = black
```

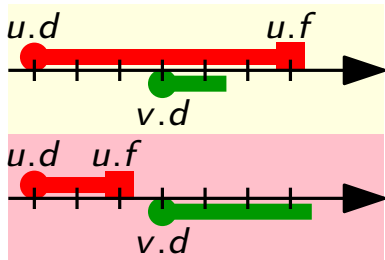
**Satz.** (Klammertheorem)

Nach  $\text{DFS}(G)$  gilt für  $\{u, v\} \in \binom{V}{2}$  genau eine der Bedingungen

- (i) Besuchsintervalle disjunkt und Baumkanten enthalten weder  $u$ - $v$ - noch  $v$ - $u$ -Weg.
- (ii)  $[u.d, u.f] \subset [v.d, v.f]$  und Baumkanten enthalten  $v$ - $u$ -Weg.
- (iii) Wie (ii), nur umgekehrt.

**Beweis.** Wir betrachten zwei Fälle.

1. Fall:  $u.d < v.d$ . ✓ 2. Fall:  $v.d < u.d$ . Symmetrisch!



A)  $v.d < u.f$ . ✓

B)  $u.f < v.d$ . ✓

Laut Code gilt außerdem  $u.d < u.f < v.d < v.f$

$$\Rightarrow [u.d, u.f] \cap [v.d, v.f] = \emptyset$$

$\Rightarrow$  Keiner der beiden Knoten wurde entdeckt, während der andere noch grau war, d.h. keiner Nachf. des anderen. (i)  $\uparrow\uparrow$



# Tiefensuche – Analyse

```

DFSVisit(Graph G, Vertex u)
  time = time + 1
  u.d = time; u.color = gray
  foreach v ∈ Adj[u] do
    if v.color == white then
      v.π = u; DFSVisit(G, v)
  time = time + 1
  u.f = time; u.color = black
    
```

**Satz.** (Klammertheorem)

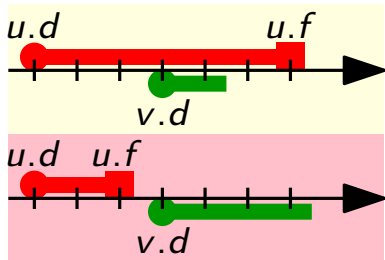
Nach  $\text{DFS}(G)$  gilt für  $\{u, v\} \in \binom{V}{2}$  genau eine der Bedingungen

- (i) Besuchsintervalle disjunkt und Baumkanten enthalten weder  $u$ - $v$ - noch  $v$ - $u$ -Weg.
- (ii)  $[u.d, u.f] \subset [v.d, v.f]$  und Baumkanten enthalten  $v$ - $u$ -Weg.
- (iii) Wie (ii), nur umgekehrt.

**Beweis.** Wir betrachten zwei Fälle.

1. Fall:  $u.d < v.d$ . ✓ 2. Fall:  $v.d < u.d$ . Symmetrisch!

(Vertausche im Beweis  $u \leftrightarrow v$ .)



A)  $v.d < u.f$ . ✓

B)  $u.f < v.d$ . ✓

Laut Code gilt außerdem  $u.d < u.f < v.d < v.f$

$$\Rightarrow [u.d, u.f] \cap [v.d, v.f] = \emptyset$$

$\Rightarrow$  Keiner der beiden Knoten wurde entdeckt, während der andere noch grau war, d.h. keiner Nachf. des anderen. (i) ↑↑

# Tiefensuche – Analyse

```

DFSVisit(Graph G, Vertex u)
  time = time + 1
  u.d = time; u.color = gray
  foreach v ∈ Adj[u] do
    if v.color == white then
      v.π = u; DFSVisit(G, v)
  time = time + 1
  u.f = time; u.color = black
    
```

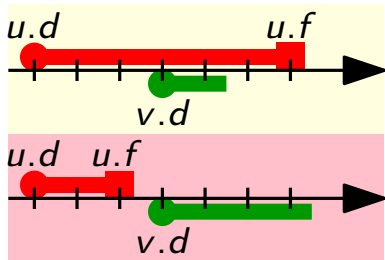
**Satz.** (Klammertheorem)

Nach  $\text{DFS}(G)$  gilt für  $\{u, v\} \in \binom{V}{2}$  genau eine der Bedingungen

- (i) Besuchsintervalle disjunkt und Baumkanten enthalten weder  $u$ - $v$ - noch  $v$ - $u$ -Weg.
- (ii)  $[u.d, u.f] \subset [v.d, v.f]$  und Baumkanten enthalten  $v$ - $u$ -Weg.
- (iii) Wie (ii), nur umgekehrt.

**Beweis.** Wir betrachten zwei Fälle.

1. Fall:  $u.d < v.d$ . ✓ 2. Fall:  $v.d < u.d$ . Symmetrisch! ✓  
 (Vertausche im Beweis  $u \leftrightarrow v$ .)



A)  $v.d < u.f$ . ✓

B)  $u.f < v.d$ . ✓

Laut Code gilt außerdem  $u.d < u.f < v.d < v.f$

$$\Rightarrow [u.d, u.f] \cap [v.d, v.f] = \emptyset$$

$\Rightarrow$  Keiner der beiden Knoten wurde entdeckt, während der andere noch grau war, d.h. keiner Nachf. des anderen. (i) ↑↑

# Tiefensuche in ungerichteten Graphen

**Satz.**  $G$  ungerichtet  
 $\Rightarrow G$  hat nur Baum- und Rückwärtskanten.

# Tiefensuche in ungerichteten Graphen

**Satz.**  $G$  ungerichtet  
 $\Rightarrow G$  hat nur Baum- und Rückwärtskanten.

*Beweis.*

# Tiefensuche in ungerichteten Graphen

**Satz.**  $G$  ungerichtet  
 $\Rightarrow G$  hat nur Baum- und Rückwärtskanten.

*Beweis.* Sei  $uv$  (kurz für  $\{u, v\}$ ) eine beliebige Kante von  $G$ .

# Tiefensuche in ungerichteten Graphen

**Satz.**  $G$  ungerichtet  
 $\Rightarrow G$  hat nur Baum- und Rückwärtskanten.

*Beweis.* Sei  $uv$  (kurz für  $\{u, v\}$ ) eine beliebige Kante von  $G$ .  
O.B.d.A. gilt  $u.d < v.d$ .

# Tiefensuche in ungerichteten Graphen

**Satz.**  $G$  ungerichtet  
 $\Rightarrow G$  hat nur Baum- und Rückwärtskanten.

*Beweis.* Sei  $uv$  (kurz für  $\{u, v\}$ ) eine beliebige Kante von  $G$ .  
O.B.d.A. gilt  $u.d < v.d$ .  
Dann entdeckt DFS  $v$  und färbt  $v$  schwarz,  
bevor  $u$  schwarz gefärbt wird

# Tiefensuche in ungerichteten Graphen

**Satz.**  $G$  ungerichtet  
 $\Rightarrow G$  hat nur Baum- und Rückwärtskanten.

*Beweis.* Sei  $uv$  (kurz für  $\{u, v\}$ ) eine beliebige Kante von  $G$ .  
O.B.d.A. gilt  $u.d < v.d$ .

Dann entdeckt DFS  $v$  und färbt  $v$  schwarz,  
bevor  $u$  schwarz gefärbt wird (da  $v \in \text{Adj}[u]$ ).

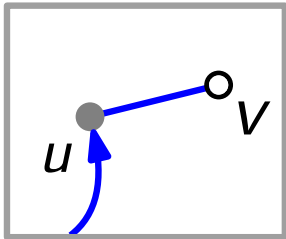


# Tiefensuche in ungerichteten Graphen

**Satz.**  $G$  ungerichtet  
 $\Rightarrow G$  hat nur Baum- und Rückwärtskanten.

**Beweis.** Sei  $uv$  (kurz für  $\{u, v\}$ ) eine beliebige Kante von  $G$ .  
O.B.d.A. gilt  $u.d < v.d$ .

Dann entdeckt DFS  $v$  und färbt  $v$  schwarz,  
bevor  $u$  schwarz gefärbt wird (da  $v \in \text{Adj}[u]$ ).



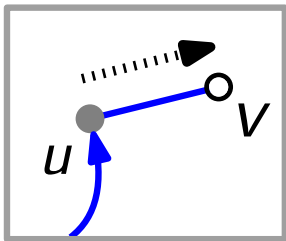
- Falls DFS  $uv$  zum ersten Mal von  $u$  nach  $v$  überschreitet, ist  $v$  zu diesem Zeitpunkt *weiss*.

# Tiefensuche in ungerichteten Graphen

**Satz.**  $G$  ungerichtet  
 $\Rightarrow G$  hat nur Baum- und Rückwärtskanten.

*Beweis.* Sei  $uv$  (kurz für  $\{u, v\}$ ) eine beliebige Kante von  $G$ .  
O.B.d.A. gilt  $u.d < v.d$ .

Dann entdeckt DFS  $v$  und färbt  $v$  schwarz,  
bevor  $u$  schwarz gefärbt wird (da  $v \in \text{Adj}[u]$ ).



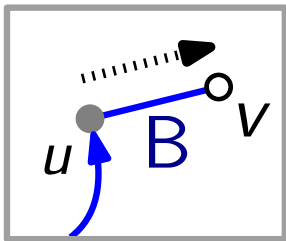
- Falls DFS  $uv$  zum ersten Mal von  $u$  nach  $v$  überschreitet, ist  $v$  zu diesem Zeitpunkt *weiss*.

# Tiefensuche in ungerichteten Graphen

**Satz.**  $G$  ungerichtet  
 $\Rightarrow G$  hat nur Baum- und Rückwärtskanten.

*Beweis.* Sei  $uv$  (kurz für  $\{u, v\}$ ) eine beliebige Kante von  $G$ .  
 O.B.d.A. gilt  $u.d < v.d$ .

Dann entdeckt DFS  $v$  und färbt  $v$  schwarz,  
 bevor  $u$  schwarz gefärbt wird (da  $v \in \text{Adj}[u]$ ).



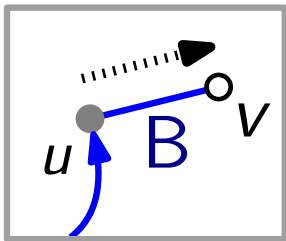
- Falls DFS  $uv$  zum ersten Mal von  $u$  nach  $v$  überschreitet, ist  $v$  zu diesem Zeitpunkt *weiss*. Dann ist  $uv$  Baumkante.

# Tiefensuche in ungerichteten Graphen

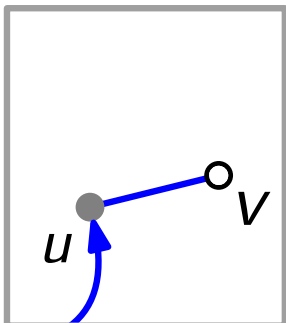
**Satz.**  $G$  ungerichtet  
 $\Rightarrow G$  hat nur Baum- und Rückwärtskanten.

*Beweis.* Sei  $uv$  (kurz für  $\{u, v\}$ ) eine beliebige Kante von  $G$ .  
 O.B.d.A. gilt  $u.d < v.d$ .

Dann entdeckt DFS  $v$  und färbt  $v$  schwarz,  
 bevor  $u$  schwarz gefärbt wird (da  $v \in \text{Adj}[u]$ ).



- Falls DFS  $uv$  zum ersten Mal von  $u$  nach  $v$  überschreitet, ist  $v$  zu diesem Zeitpunkt *weiss*. Dann ist  $uv$  Baumkante.



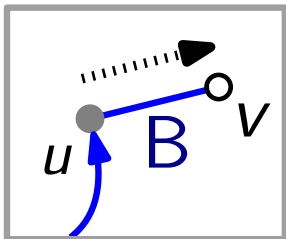
- Andernfalls wird  $uv$  zum ersten Mal von  $v$  nach  $u$  überschritten.

# Tiefensuche in ungerichteten Graphen

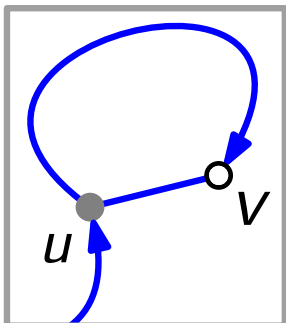
**Satz.**  $G$  ungerichtet  
 $\Rightarrow G$  hat nur Baum- und Rückwärtskanten.

*Beweis.* Sei  $uv$  (kurz für  $\{u, v\}$ ) eine beliebige Kante von  $G$ .  
 O.B.d.A. gilt  $u.d < v.d$ .

Dann entdeckt DFS  $v$  und färbt  $v$  schwarz,  
 bevor  $u$  schwarz gefärbt wird (da  $v \in \text{Adj}[u]$ ).



- Falls DFS  $uv$  zum ersten Mal von  $u$  nach  $v$  überschreitet, ist  $v$  zu diesem Zeitpunkt *weiss*. Dann ist  $uv$  Baumkante.



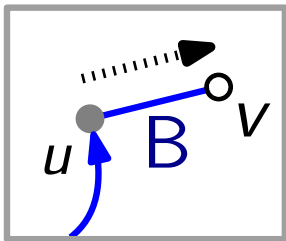
- Andernfalls wird  $uv$  zum ersten Mal von  $v$  nach  $u$  überschritten.

# Tiefensuche in ungerichteten Graphen

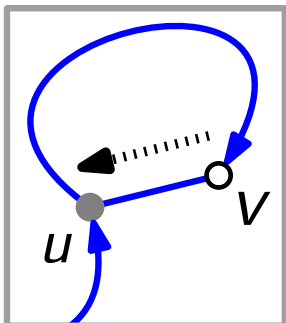
**Satz.**  $G$  ungerichtet  
 $\Rightarrow G$  hat nur Baum- und Rückwärtskanten.

*Beweis.* Sei  $uv$  (kurz für  $\{u, v\}$ ) eine beliebige Kante von  $G$ .  
 O.B.d.A. gilt  $u.d < v.d$ .

Dann entdeckt DFS  $v$  und färbt  $v$  schwarz,  
 bevor  $u$  schwarz gefärbt wird (da  $v \in \text{Adj}[u]$ ).



- Falls DFS  $uv$  zum ersten Mal von  $u$  nach  $v$  überschreitet, ist  $v$  zu diesem Zeitpunkt *weiss*. Dann ist  $uv$  Baumkante.



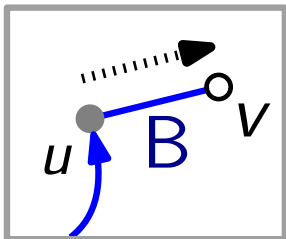
- Andernfalls wird  $uv$  zum ersten Mal von  $v$  nach  $u$  überschritten.

# Tiefensuche in ungerichteten Graphen

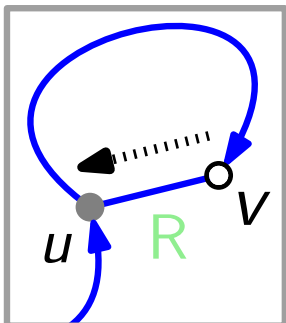
**Satz.**  $G$  ungerichtet  
 $\Rightarrow G$  hat nur Baum- und Rückwärtskanten.

**Beweis.** Sei  $uv$  (kurz für  $\{u, v\}$ ) eine beliebige Kante von  $G$ .  
 O.B.d.A. gilt  $u.d < v.d$ .

Dann entdeckt DFS  $v$  und färbt  $v$  schwarz,  
 bevor  $u$  schwarz gefärbt wird (da  $v \in \text{Adj}[u]$ ).



- Falls DFS  $uv$  zum ersten Mal von  $u$  nach  $v$  überschreitet, ist  $v$  zu diesem Zeitpunkt *weiss*. Dann ist  $uv$  Baumkante.



- Andernfalls wird  $uv$  zum ersten Mal von  $v$  nach  $u$  überschritten. Dann ist  $uv$  R-Kante, da  $u$  dann schon (und immer noch) *grau* ist.



# Ablaufplanung

Unterhose

Socken

Hose

Schuhe

Gürtel

Uhr

Schal

T-Shirt

Anorak

Pulli



# Ablaufplanung

Kante bedeutet:  
Unterhose *vor*  
Hose anziehen!

Unterhose

Socken

Hose

Schuhe

Gürtel

Uhr

Schal

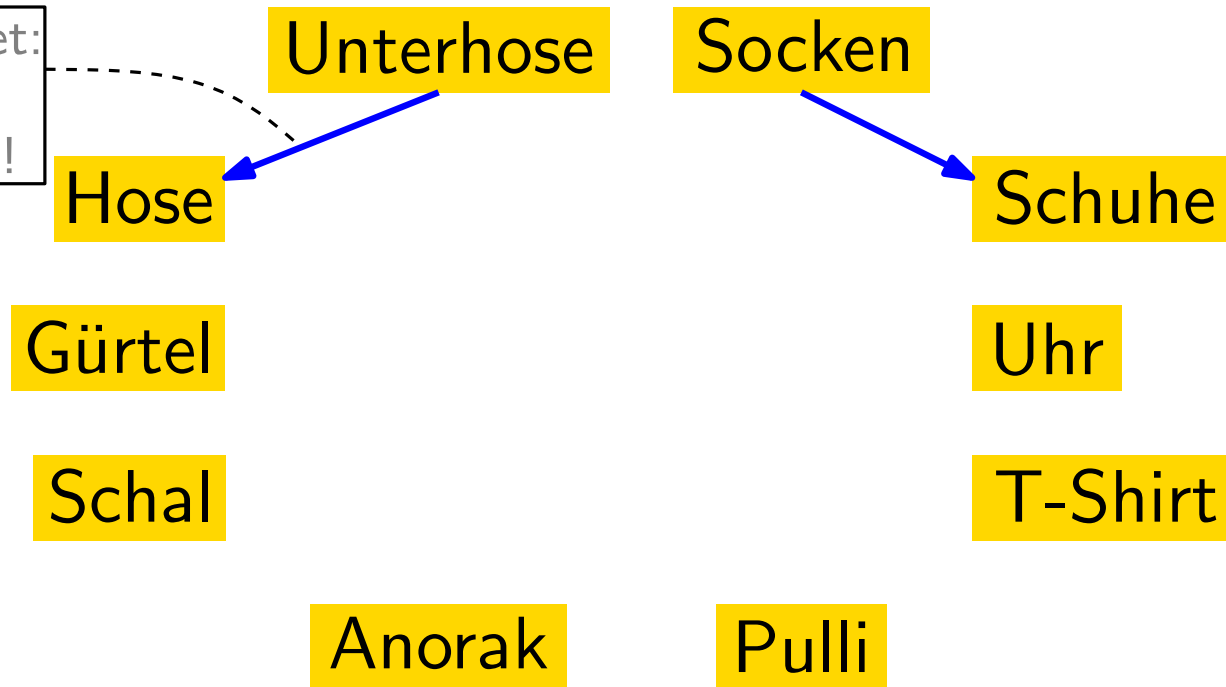
T-Shirt

Anorak

Pulli

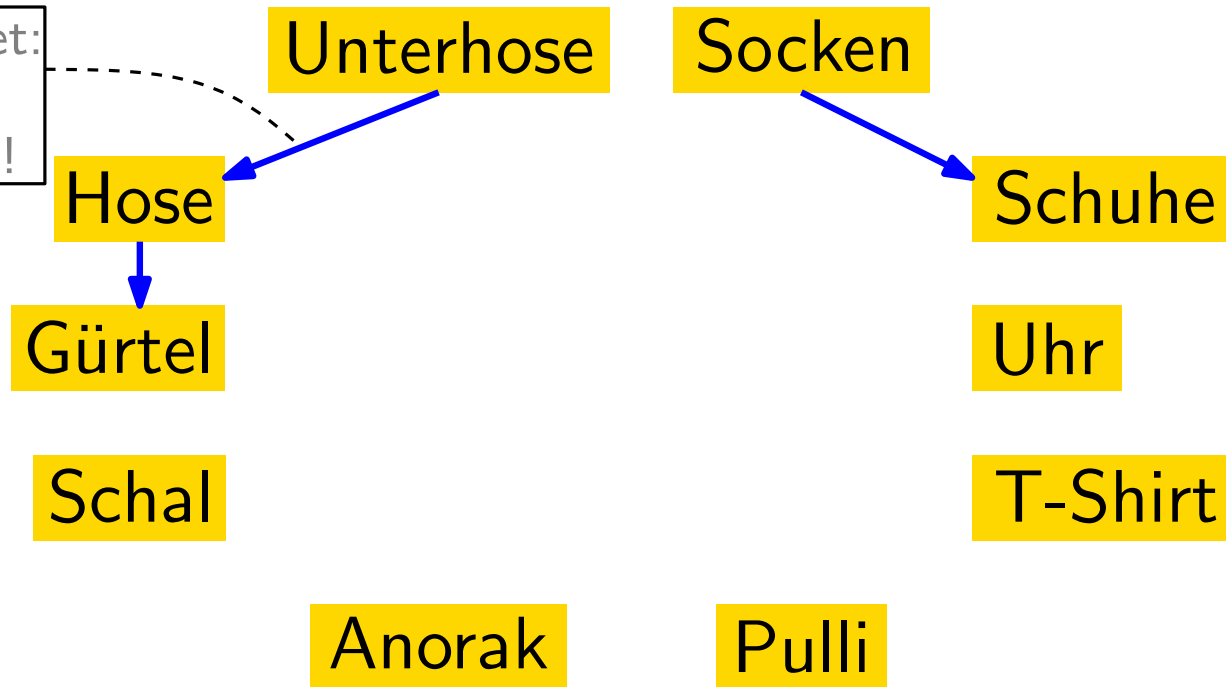
# Ablaufplanung

Kante bedeutet:  
Unterhose *vor*  
Hose anziehen!



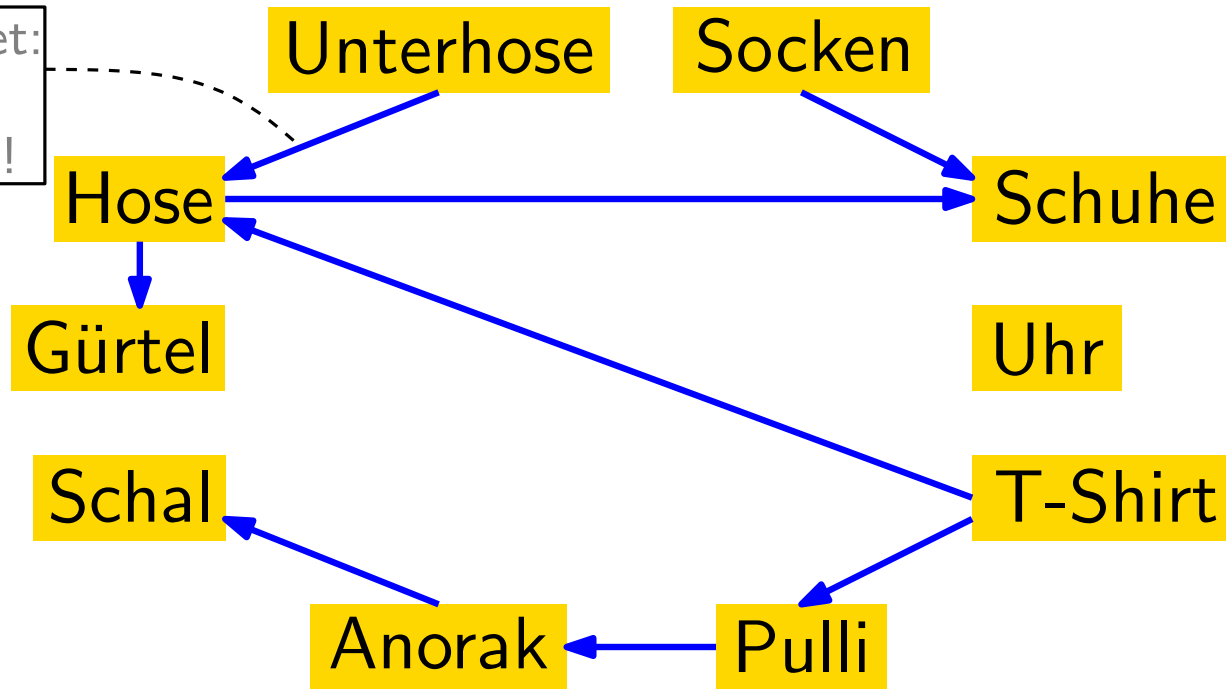
# Ablaufplanung

Kante bedeutet:  
Unterhose *vor*  
Hose anziehen!

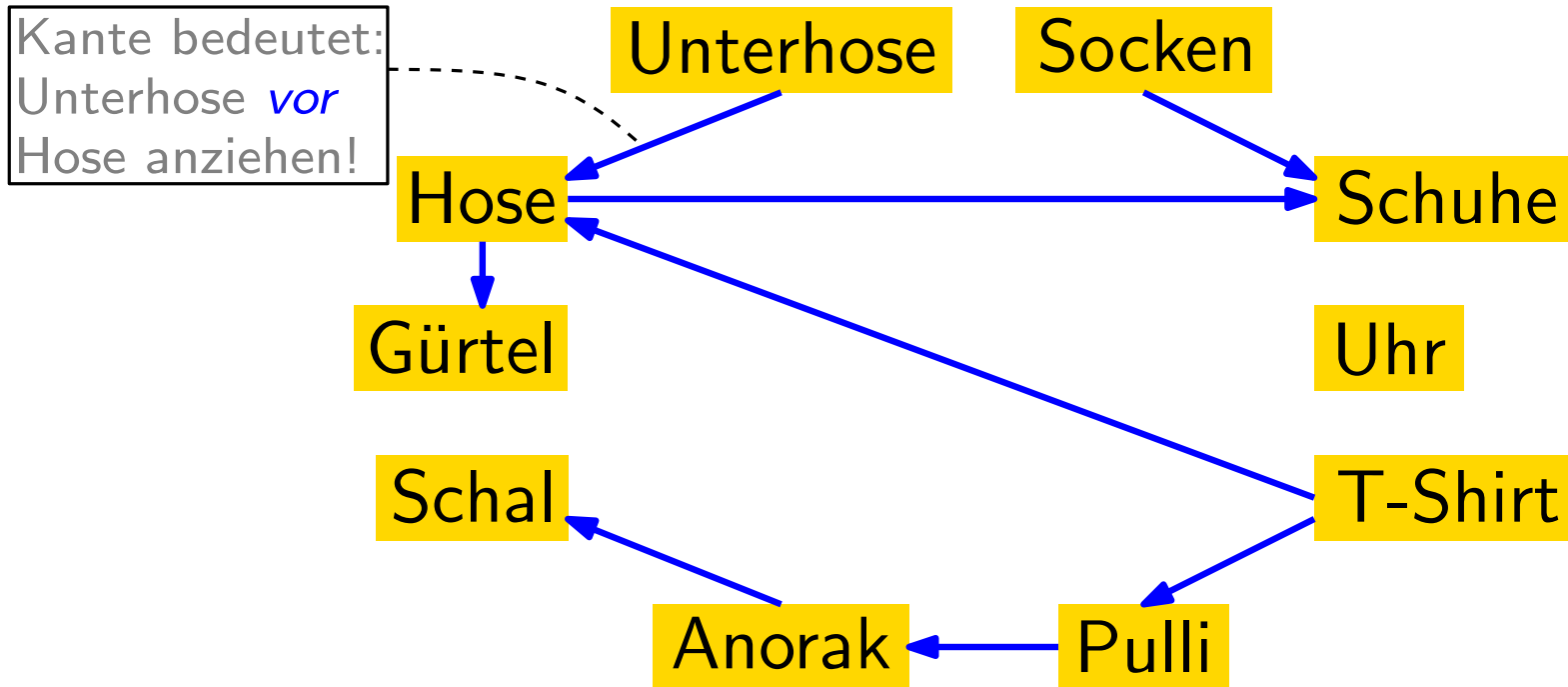


# Ablaufplanung

Kante bedeutet:  
Unterhose *vor*  
Hose anziehen!

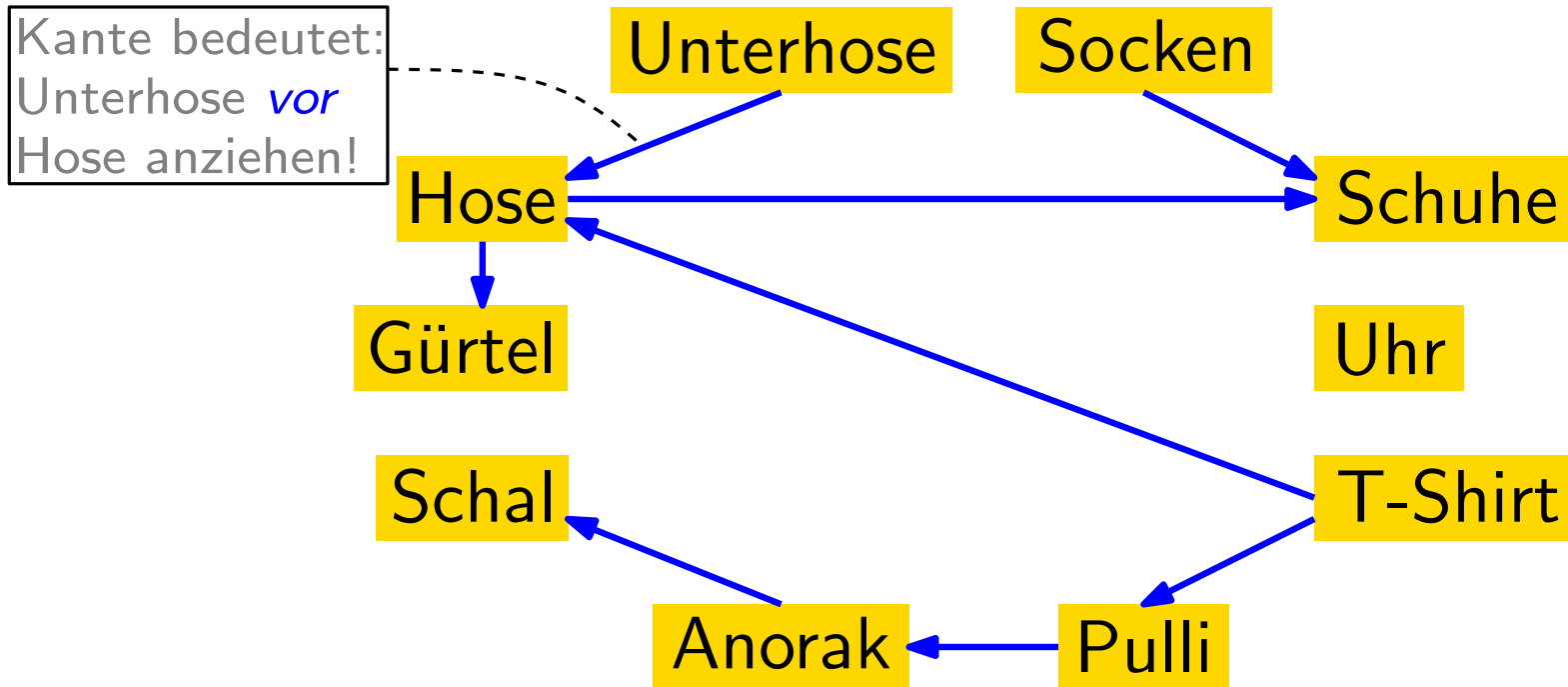


# Ablaufplanung



**Aufgabe:** Finde Ablaufplan –  
d.h. Reihenfolge der Knoten, so dass alle Einschränkungen erfüllt sind (z.B. T-Shirt vor Pulli).

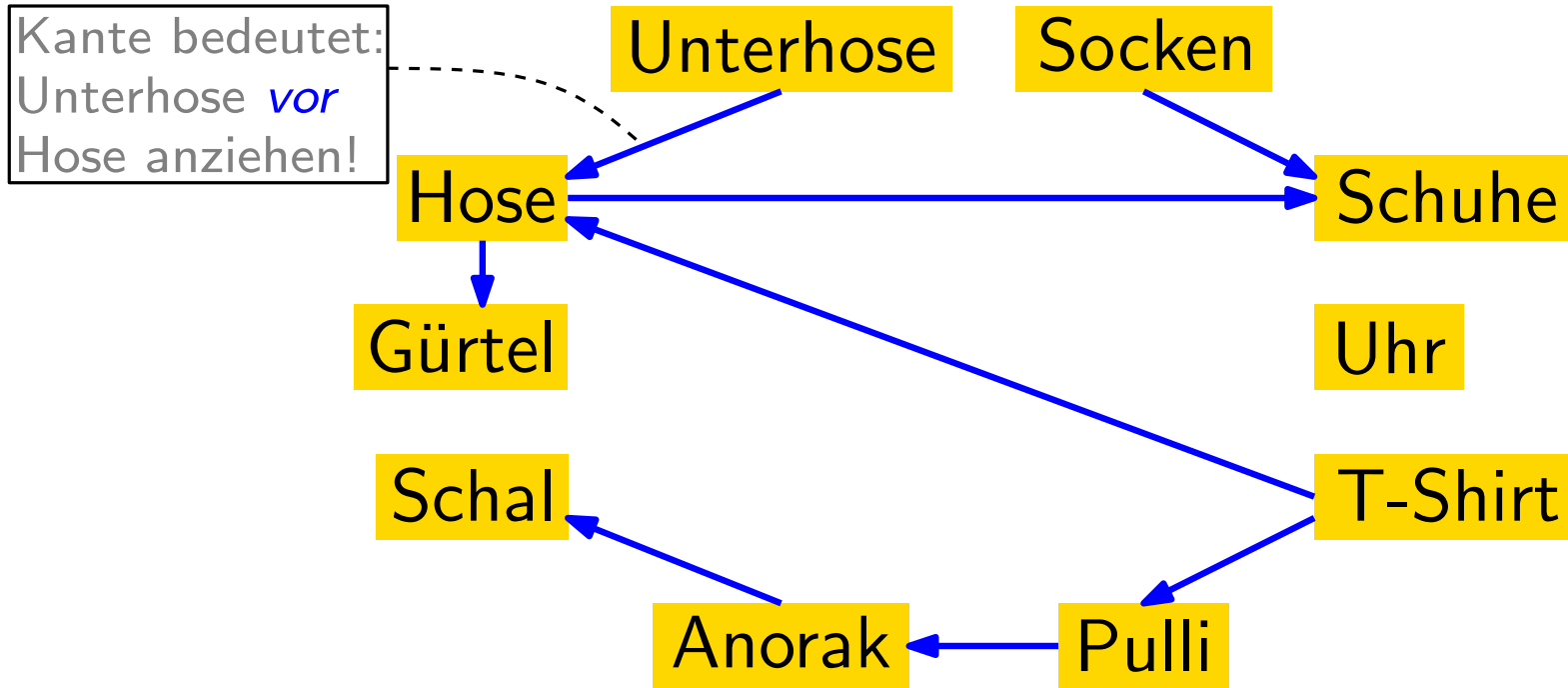
# Ablaufplanung



**Aufgabe:** Finde Ablaufplan –  
d.h. Reihenfolge der Knoten, so dass alle Einschränkungen erfüllt sind (z.B. T-Shirt vor Pulli).

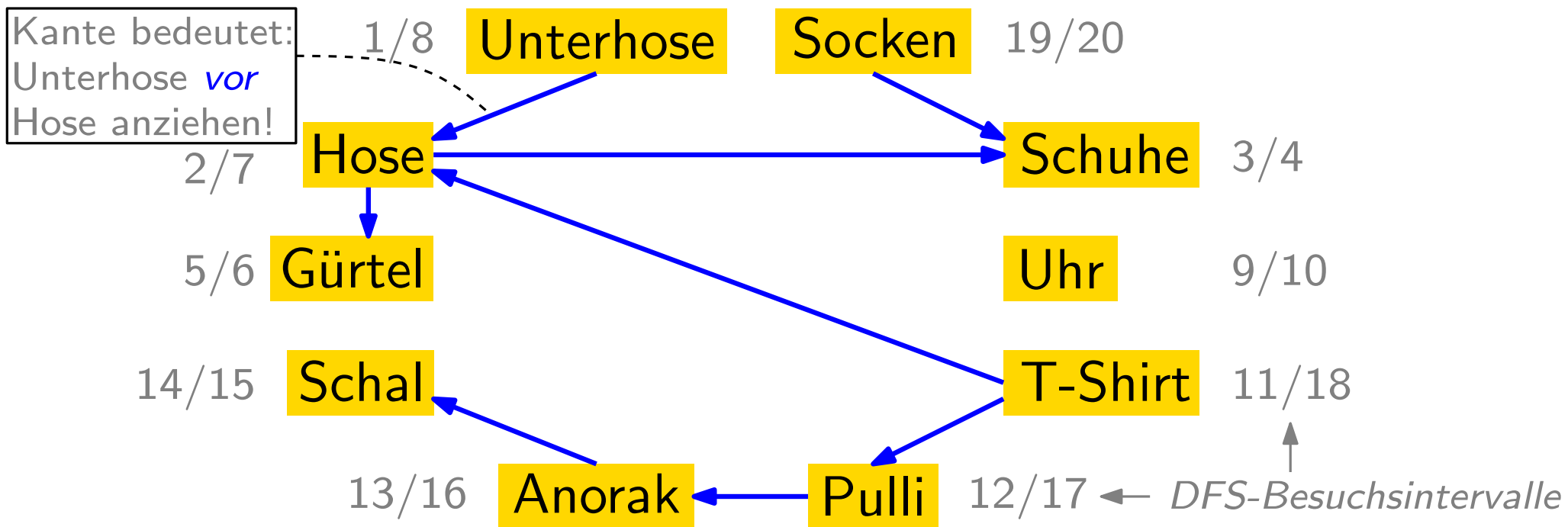
**Topologische Sortierung:** Lineare Ordnung der Knoten, so dass aus  $(u, v) \in E$  folgt:  $u$  kommt vor  $v$ .

# Ablaufplanung



**Idee:** Nutze Tiefensuche!

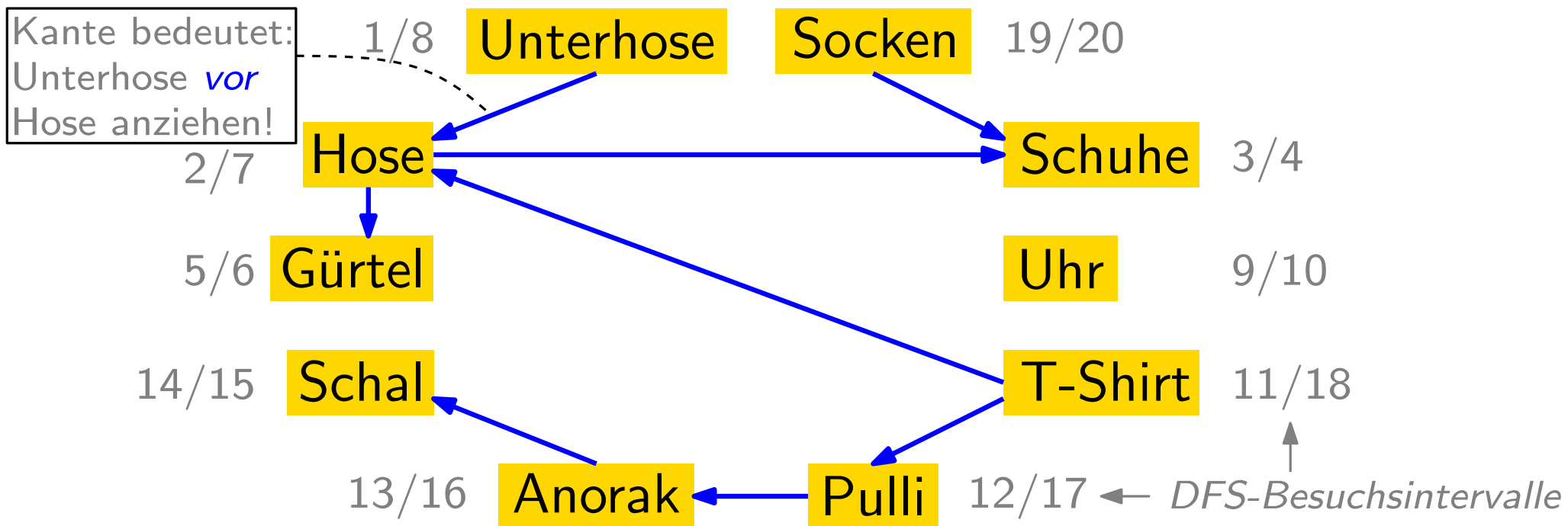
# Ablaufplanung



**Idee:** Nutze Tiefensuche!

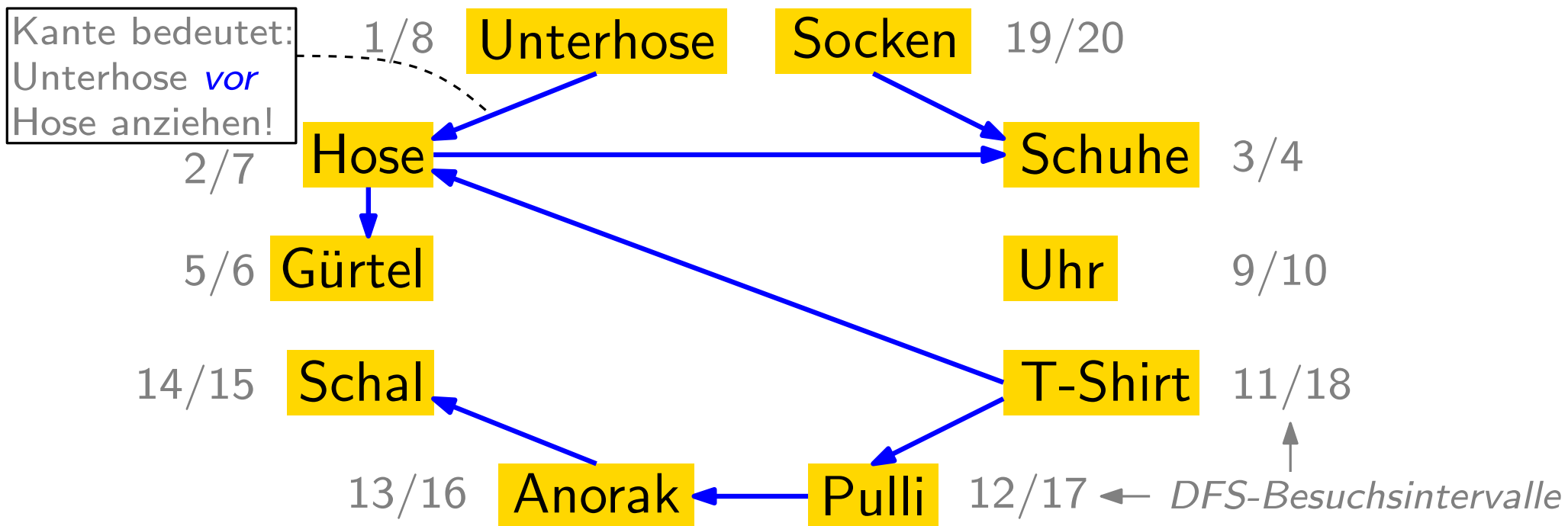


# Ablaufplanung



**Idee:** Nutze Tiefensuche!  
Sortiere Knoten nach absteigenden  $f$ -Zeiten.

# Ablaufplanung



**Idee:** Nutze Tiefensuche!  
Sortiere Knoten nach absteigenden  $f$ -Zeiten.

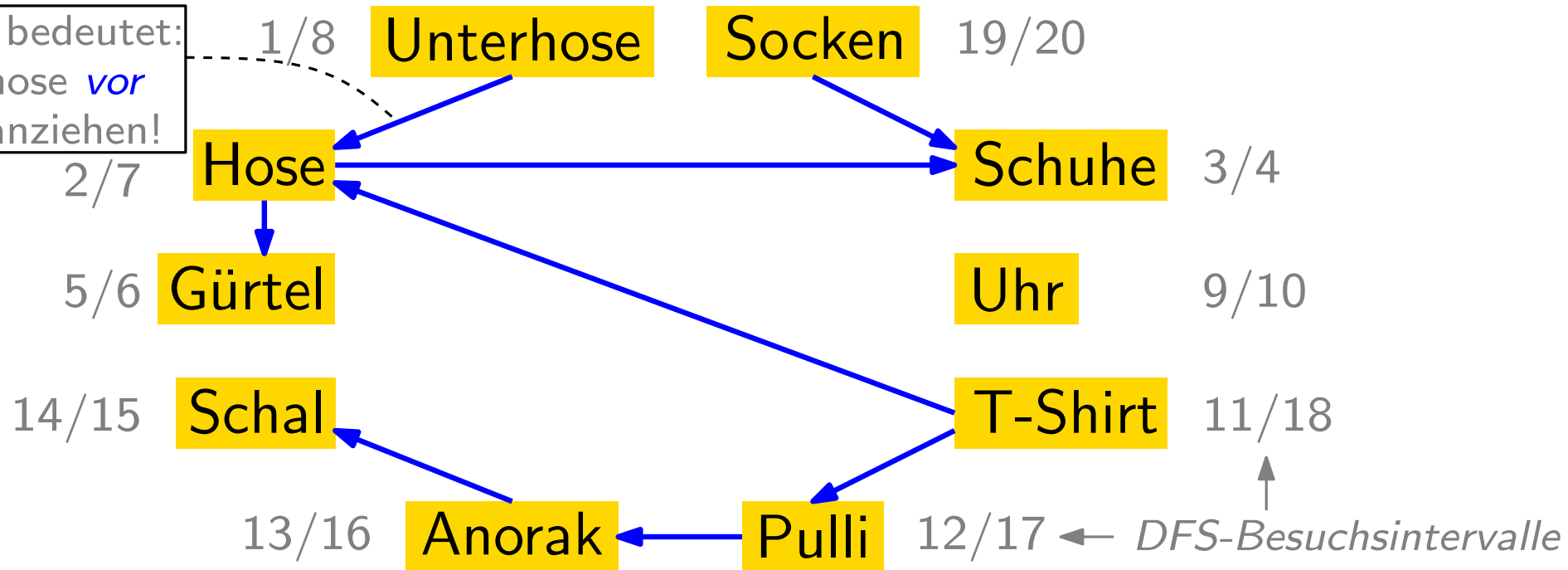
19/20

Socken

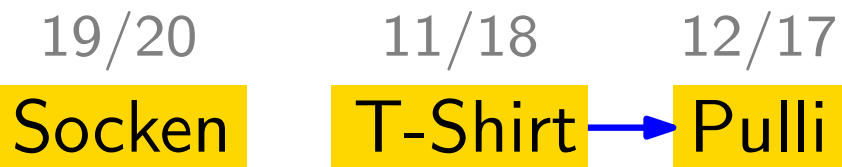


# Ablaufplanung

Kante bedeutet:  
Unterhose *vor*  
Hose anziehen!



**Idee:** Nutze Tiefensuche!  
Sortiere Knoten nach absteigenden  $f$ -Zeiten.

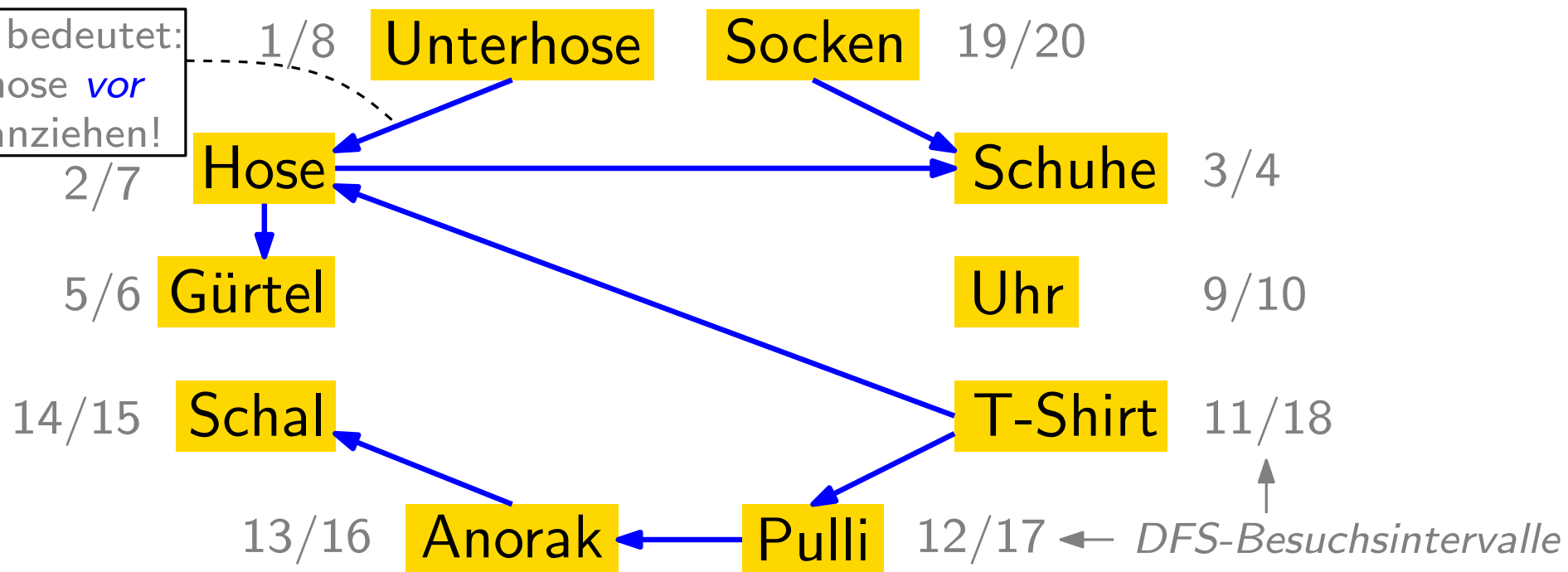




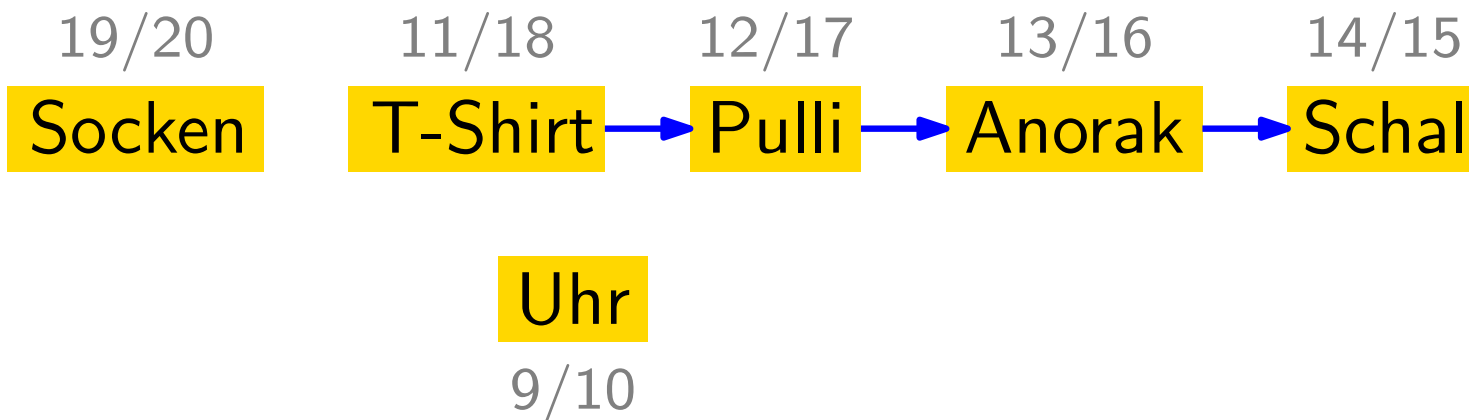


# Ablaufplanung

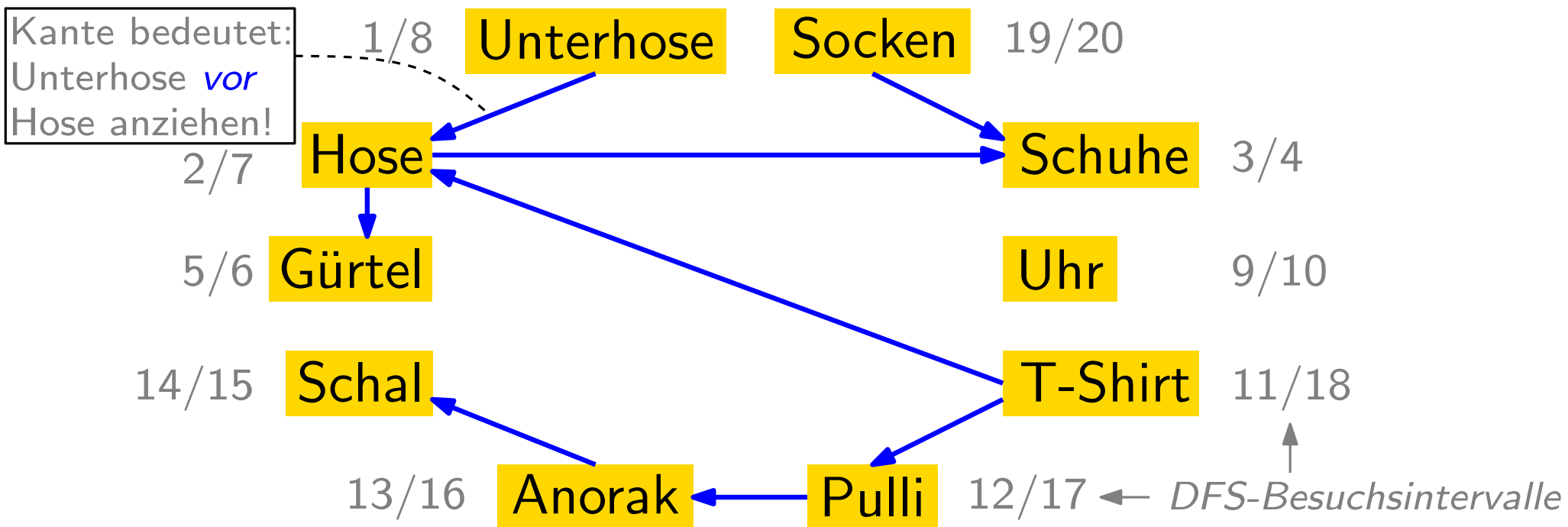
Kante bedeutet:  
Unterhose *vor*  
Hose anziehen!



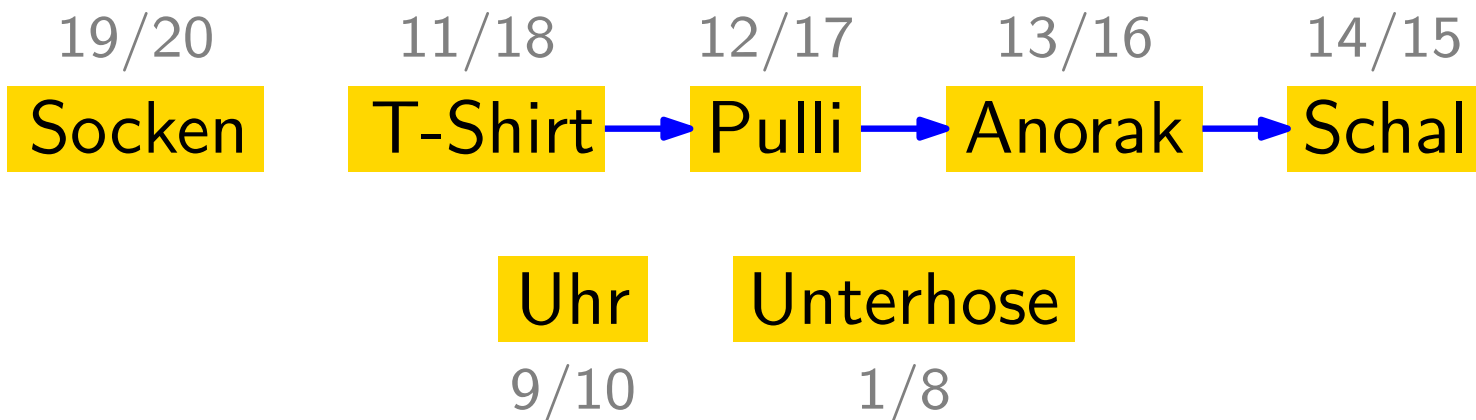
**Idee:** Nutze Tiefensuche!  
Sortiere Knoten nach absteigenden  $f$ -Zeiten.



# Ablaufplanung

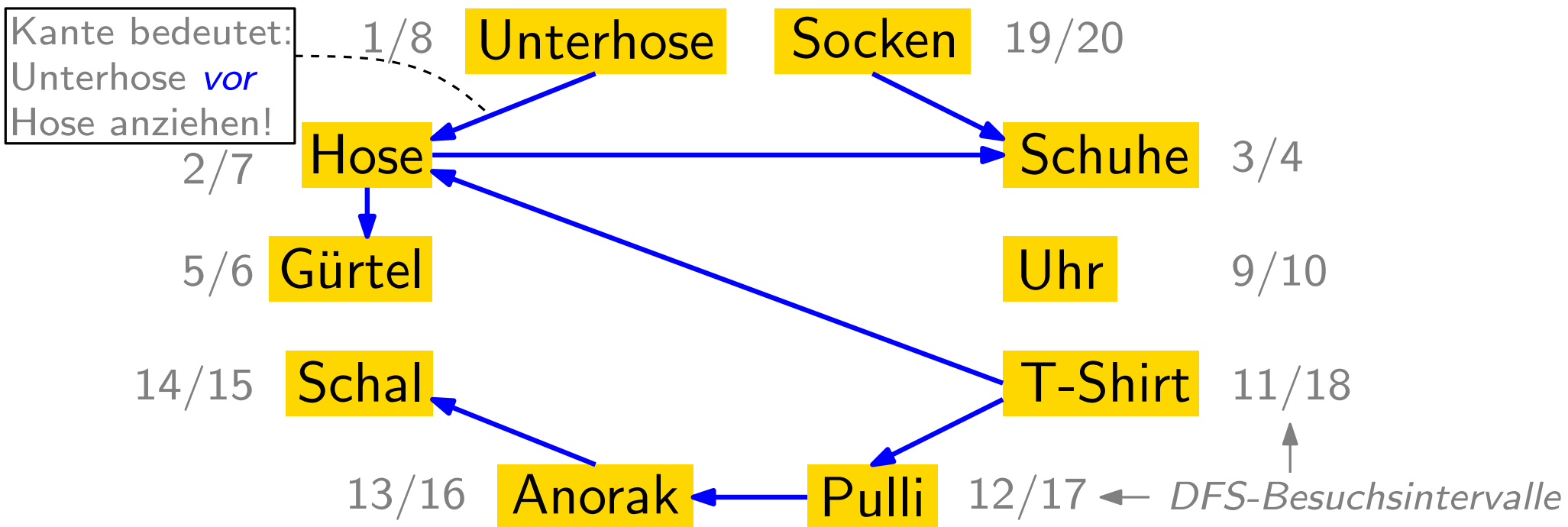


**Idee:** Nutze Tiefensuche!  
Sortiere Knoten nach absteigenden  $f$ -Zeiten.

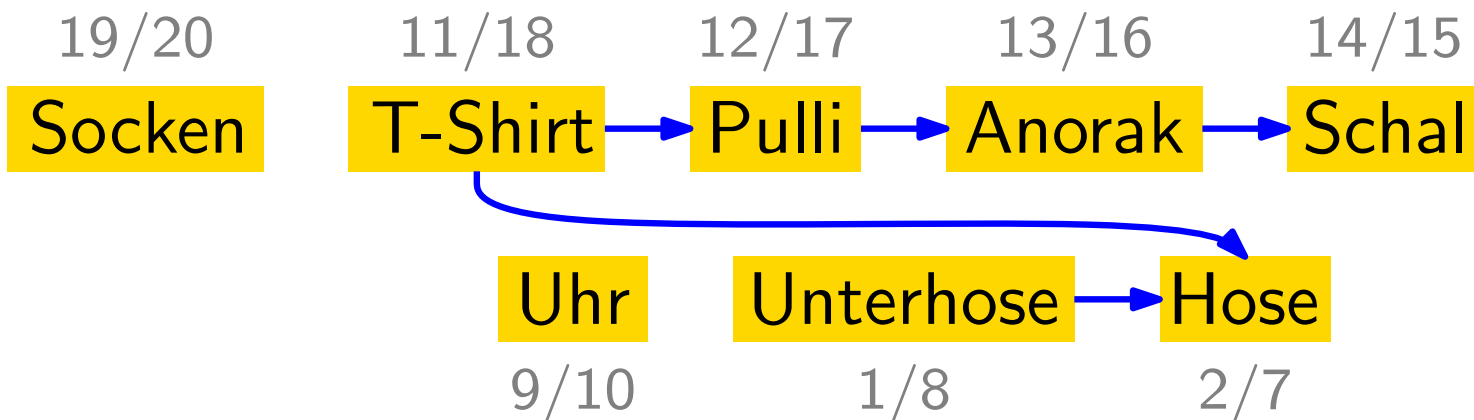




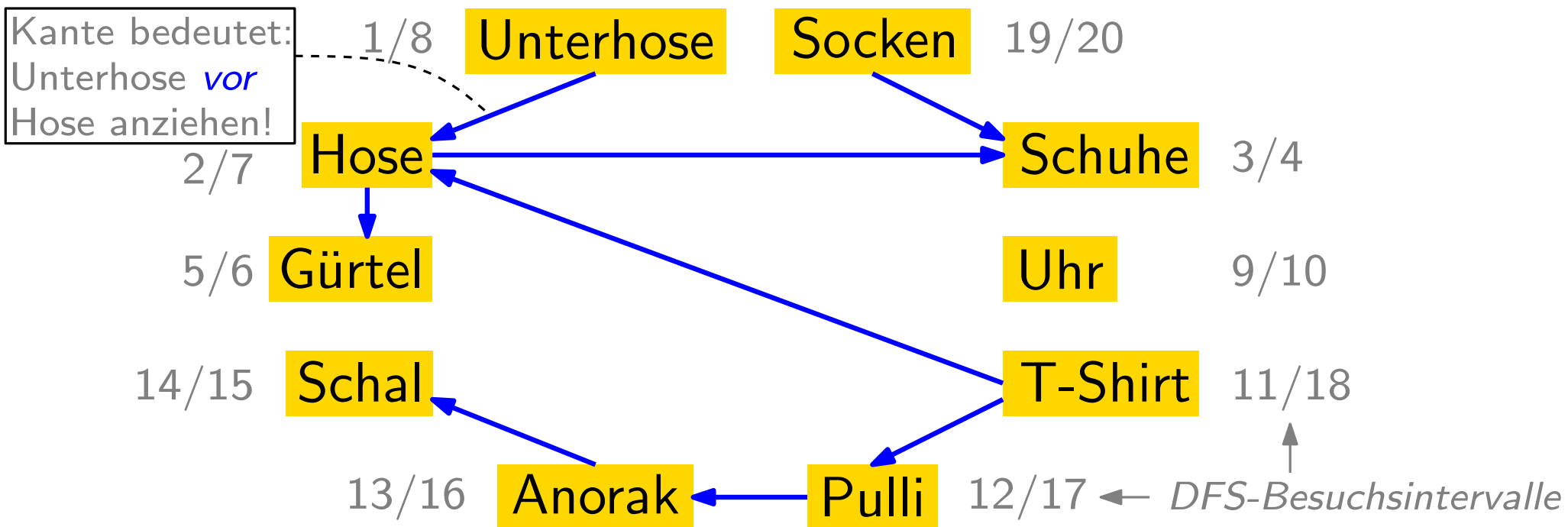
# Ablaufplanung



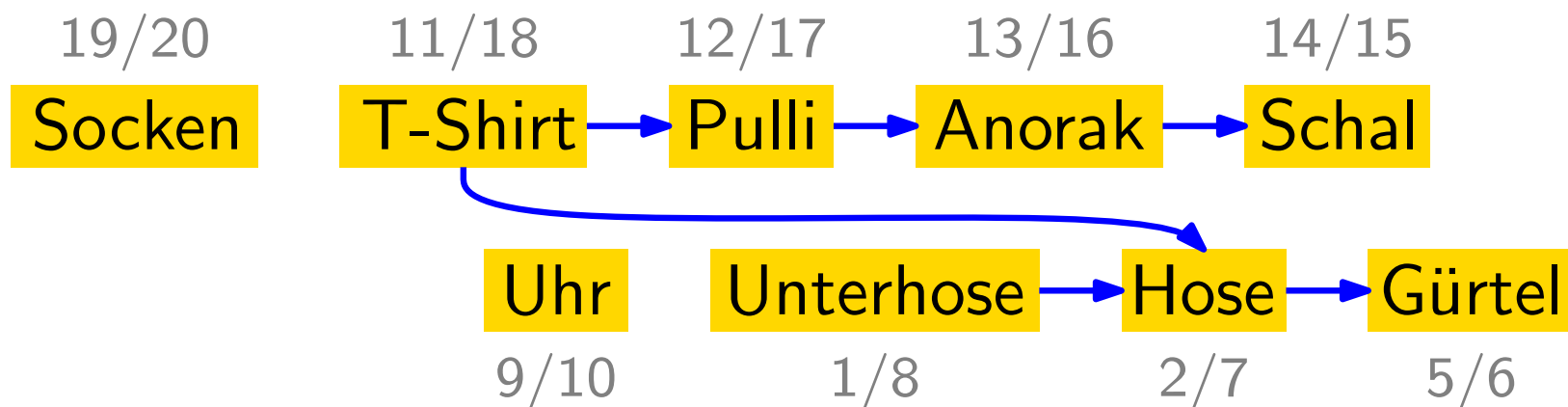
**Idee:** Nutze Tiefensuche!  
Sortiere Knoten nach absteigenden  $f$ -Zeiten.



# Ablaufplanung

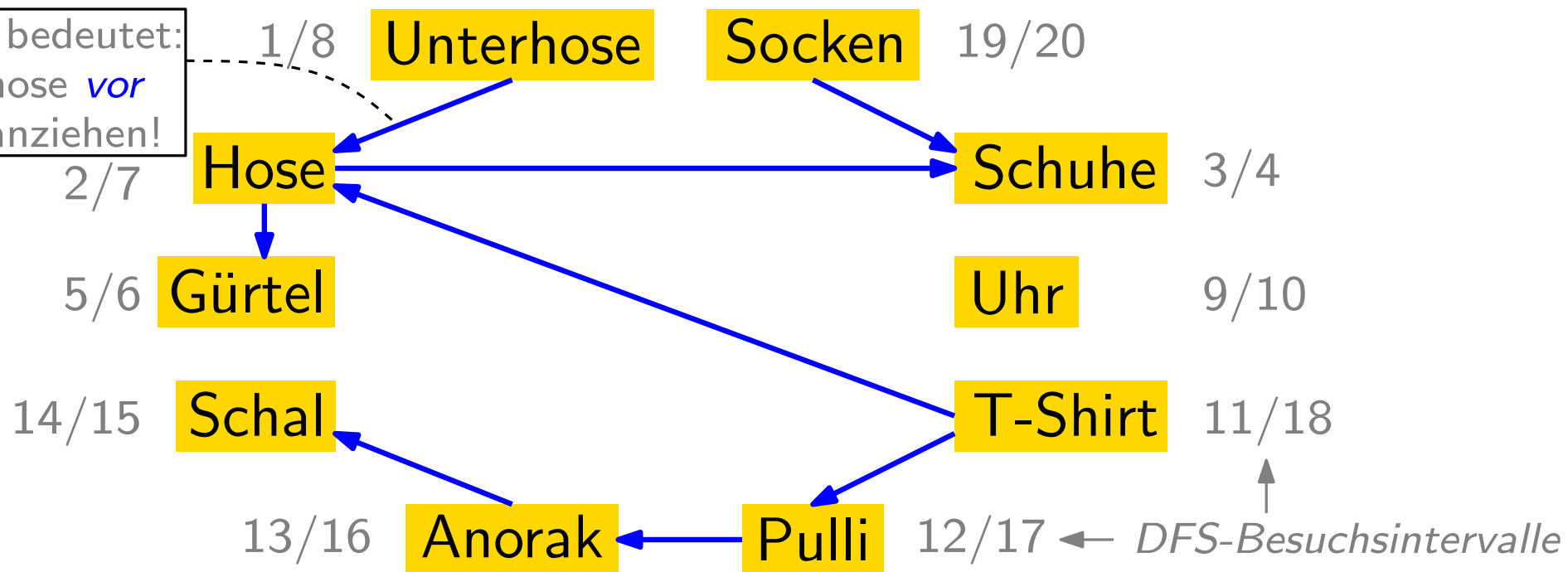


**Idee:** Nutze Tiefensuche!  
Sortiere Knoten nach absteigenden  $f$ -Zeiten.

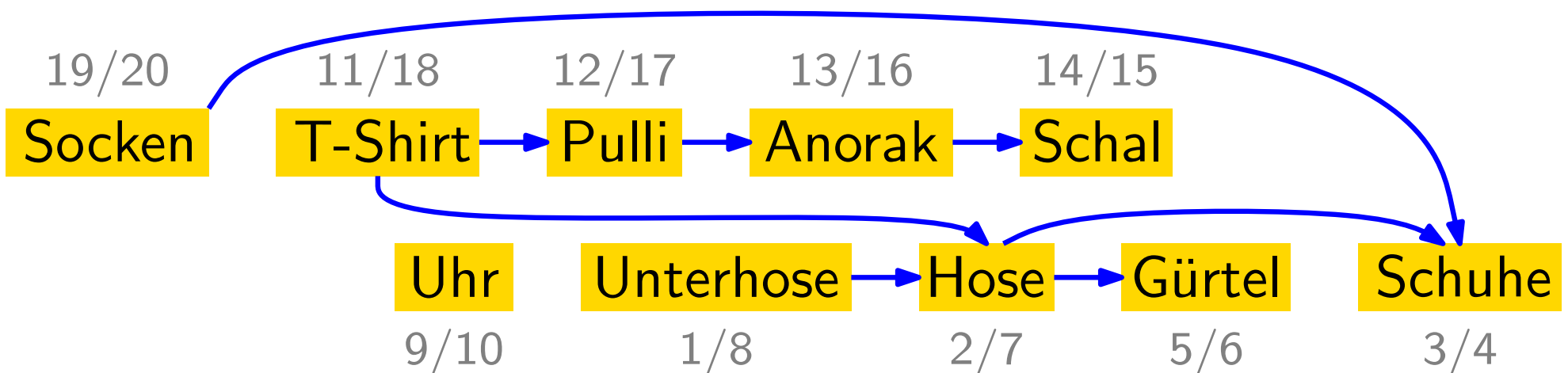


# Ablaufplanung

Kante bedeutet:  
Unterhose *vor*  
Hose anziehen!

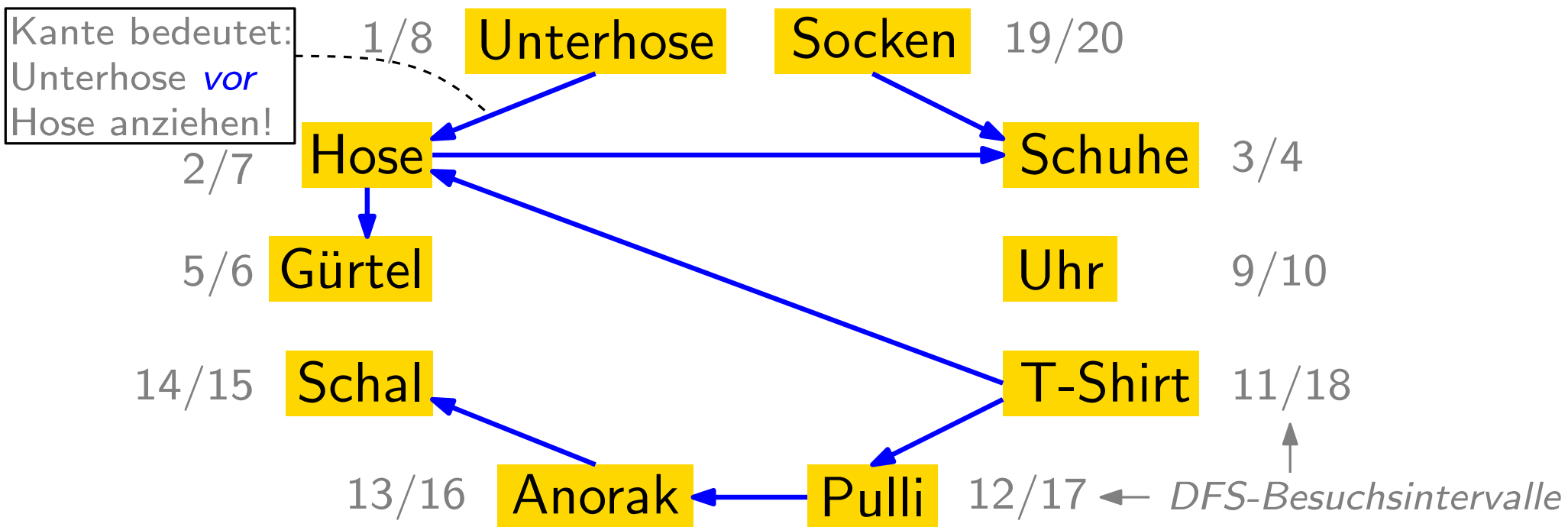


**Idee:** Nutze Tiefensuche!  
Sortiere Knoten nach absteigenden  $f$ -Zeiten.

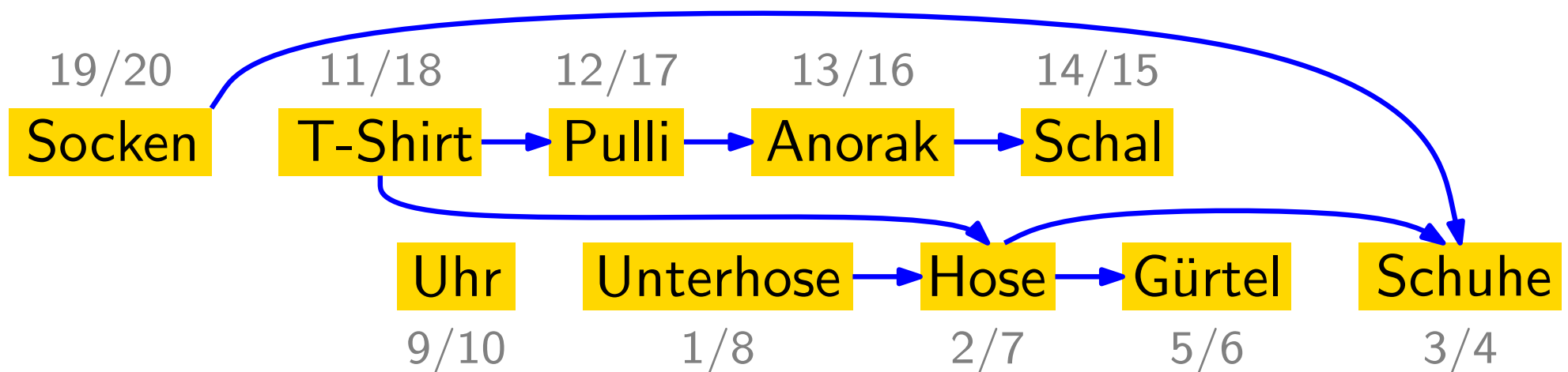




# Ablaufplanung



**Idee:** Nutze Tiefensuche!  $\Rightarrow$  Alle Kanten sind nach rechts gerichtet. Sortiere Knoten nach absteigenden  $f$ -Zeiten.



# Topologisch sortieren

*Topologische Sortierung:* Lineare Ordnung der Knoten, so dass aus  $(u, v) \in E$  folgt:  $u$  kommt vor  $v$ .

# Topologisch sortieren

*Topologische Sortierung:* Lineare Ordnung der Knoten, so dass aus  $(u, v) \in E$  folgt:  $u$  kommt vor  $v$ .

TopologicalSort(DirectedGraph  $G$ )

# Topologisch sortieren

*Topologische Sortierung:* Lineare Ordnung der Knoten, so dass aus  $(u, v) \in E$  folgt:  $u$  kommt vor  $v$ .

```
TopologicalSort(DirectedGraph  $G$ )
```

```
   $L = \mathbf{new}$  List()
```

```
  DFS( $G$ ) mit folgender Änderung:
```

```
    Wenn ein Knoten schwarz gefärbt wird,  
    häng ihn vorne an die Liste  $L$  an.
```

```
  return  $L$ 
```



# Topologisch sortieren

*Topologische Sortierung:* Lineare Ordnung der Knoten, so dass aus  $(u, v) \in E$  folgt:  $u$  kommt vor  $v$ .

```
TopologicalSort(DirectedGraph  $G$ )
```

```
   $L = \mathbf{new}$  List()
```

```
  DFS( $G$ ) mit folgender Änderung:
```

```
    Wenn ein Knoten schwarz gefärbt wird,  
    häng ihn vorne an die Liste  $L$  an.
```

```
  return  $L$ 
```

**Laufzeit?**

# Topologisch sortieren

*Topologische Sortierung:* Lineare Ordnung der Knoten, so dass aus  $(u, v) \in E$  folgt:  $u$  kommt vor  $v$ .

```
TopologicalSort(DirectedGraph  $G$ )
```

```
   $L = \mathbf{new}$  List()
```

```
  DFS( $G$ ) mit folgender Änderung:
```

```
    Wenn ein Knoten schwarz gefärbt wird,  
    häng ihn vorne an die Liste  $L$  an.
```

```
  return  $L$ 
```

**Laufzeit?**  
 $O(V + E)$

# Topologisch sortieren

*Topologische Sortierung:* Lineare Ordnung der Knoten, so dass aus  $(u, v) \in E$  folgt:  $u$  kommt vor  $v$ .

TopologicalSort(DirectedGraph  $G$ )

$L = \mathbf{new}$  List()

DFS( $G$ ) mit folgender Änderung:

Wenn ein Knoten schwarz gefärbt wird,  
häng ihn *vorne* an die Liste  $L$  an.

**return**  $L$

**Laufzeit?**

$O(V + E)$

**Korrekt?**

Wann funktioniert's?

# Topologisch sortieren

*Topologische Sortierung:* Lineare Ordnung der Knoten, so dass aus  $(u, v) \in E$  folgt:  $u$  kommt vor  $v$ .

TopologicalSort(DirectedGraph  $G$ )

$L = \mathbf{new}$  List()

DFS( $G$ ) mit folgender Änderung:

Wenn ein Knoten schwarz gefärbt wird,  
häng ihn *vorne* an die Liste  $L$  an.

**return**  $L$

**Laufzeit?**

$O(V + E)$

**Korrekt?**

Wann funktioniert's?

**Def.** Ein (gerichteter) Graph ist *kreisfrei*, wenn er keinen (gerichteten) Kreis enthält.

# Topologisch sortieren

*Topologische Sortierung:* Lineare Ordnung der Knoten, so dass aus  $(u, v) \in E$  folgt:  $u$  kommt vor  $v$ .

TopologicalSort(DirectedGraph  $G$ )

$L = \mathbf{new}$  List()

DFS( $G$ ) mit folgender Änderung:

Wenn ein Knoten schwarz gefärbt wird,  
häng ihn *vorne* an die Liste  $L$  an.

**return**  $L$

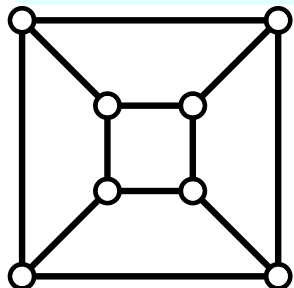
**Laufzeit?**

$O(V + E)$

**Korrekt?**

Wann funktioniert's?

**Def.** Ein (gerichteter) Graph ist *kreisfrei*, wenn er keinen (gerichteten) Kreis enthält.



# Topologisch sortieren

*Topologische Sortierung:* Lineare Ordnung der Knoten, so dass aus  $(u, v) \in E$  folgt:  $u$  kommt vor  $v$ .

TopologicalSort(DirectedGraph  $G$ )

$L = \mathbf{new}$  List()

DFS( $G$ ) mit folgender Änderung:

Wenn ein Knoten schwarz gefärbt wird,  
häng ihn *vorne* an die Liste  $L$  an.

**return**  $L$

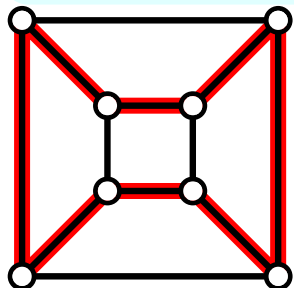
**Laufzeit?**

$O(V + E)$

**Korrekt?**

Wann funktioniert's?

**Def.** Ein (gerichteter) Graph ist *kreisfrei*, wenn er keinen (gerichteten) Kreis enthält.



# Topologisch sortieren

*Topologische Sortierung:* Lineare Ordnung der Knoten, so dass aus  $(u, v) \in E$  folgt:  $u$  kommt vor  $v$ .

TopologicalSort(DirectedGraph  $G$ )

$L = \mathbf{new}$  List()

DFS( $G$ ) mit folgender Änderung:

Wenn ein Knoten schwarz gefärbt wird,  
häng ihn *vorne* an die Liste  $L$  an.

**return**  $L$

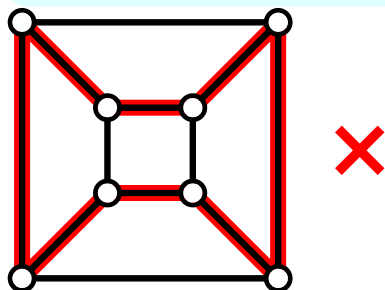
**Laufzeit?**

$O(V + E)$

**Korrekt?**

Wann funktioniert's?

**Def.** Ein (gerichteter) Graph ist *kreisfrei*, wenn er keinen (gerichteten) Kreis enthält.



# Topologisch sortieren

*Topologische Sortierung:* Lineare Ordnung der Knoten, so dass aus  $(u, v) \in E$  folgt:  $u$  kommt vor  $v$ .

TopologicalSort(DirectedGraph  $G$ )

$L = \mathbf{new}$  List()

DFS( $G$ ) mit folgender Änderung:

Wenn ein Knoten schwarz gefärbt wird,  
häng ihn *vorne* an die Liste  $L$  an.

**return**  $L$

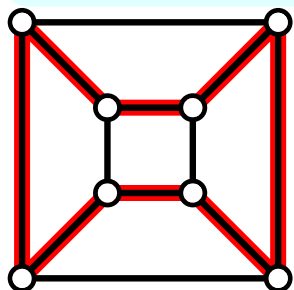
**Laufzeit?**

$O(V + E)$

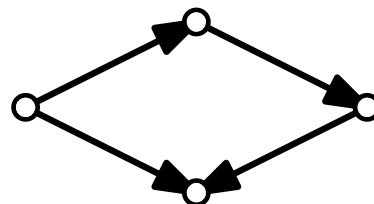
**Korrekt?**

Wann funktioniert's?

**Def.** Ein (gerichteter) Graph ist *kreisfrei*, wenn er keinen (gerichteten) Kreis enthält.



×





# Topologisch sortieren

*Topologische Sortierung:* Lineare Ordnung der Knoten, so dass aus  $(u, v) \in E$  folgt:  $u$  kommt vor  $v$ .

TopologicalSort(DirectedGraph  $G$ )

$L = \mathbf{new}$  List()

DFS( $G$ ) mit folgender Änderung:

Wenn ein Knoten schwarz gefärbt wird,  
häng ihn *vorne* an die Liste  $L$  an.

**return**  $L$

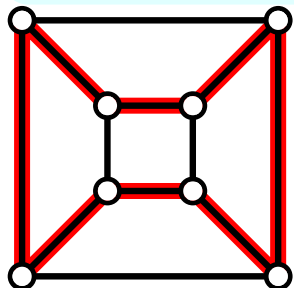
**Laufzeit?**

$O(V + E)$

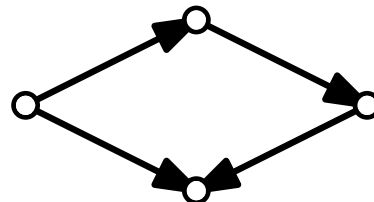
**Korrekt?**

Wann funktioniert's?

**Def.** Ein (gerichteter) Graph ist *kreisfrei*, wenn er keinen (gerichteten) Kreis enthält.



✗



✓

# Kreisfrei $\Leftrightarrow$ keine R-Kanten

**Lem.**

Ein gerichteter Graph  $G$  ist kreisfrei  
 $\Leftrightarrow$  DFS( $G$ ) liefert keine Rückwärtskanten.

# Kreisfrei $\Leftrightarrow$ keine R-Kanten

**Lem.** Ein gerichteter Graph  $G$  ist kreisfrei  
 $\Leftrightarrow$  DFS( $G$ ) liefert keine Rückwärtskanten.

*Beweis.* „ $\Rightarrow$ “

„ $\Leftarrow$ “

# Kreisfrei $\Leftrightarrow$ keine R-Kanten

**Lem.** Ein gerichteter Graph  $G$  ist kreisfrei  
 $\Leftrightarrow$  DFS( $G$ ) liefert keine Rückwärtskanten.

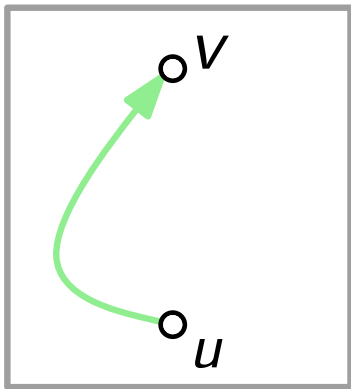
*Beweis.* „ $\Rightarrow$ “ Sei  $G$  kreisfrei.

„ $\Leftarrow$ “

# Kreisfrei $\Leftrightarrow$ keine R-Kanten

**Lem.** Ein gerichteter Graph  $G$  ist kreisfrei  
 $\Leftrightarrow$  DFS( $G$ ) liefert keine Rückwärtskanten.

**Beweis.** „ $\Rightarrow$ “ Sei  $G$  kreisfrei.



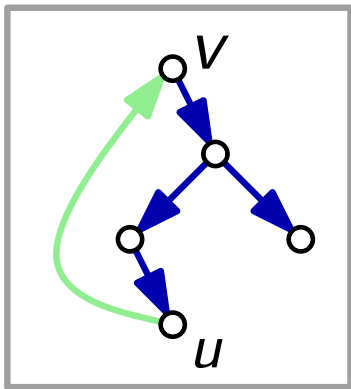
Angenommen DFS( $G$ ) liefert R-Kante  $(u, v)$ .

„ $\Leftarrow$ “

# Kreisfrei $\Leftrightarrow$ keine R-Kanten

**Lem.** Ein gerichteter Graph  $G$  ist kreisfrei  
 $\Leftrightarrow$  DFS( $G$ ) liefert keine Rückwärtskanten.

**Beweis.** „ $\Rightarrow$ “ Sei  $G$  kreisfrei.



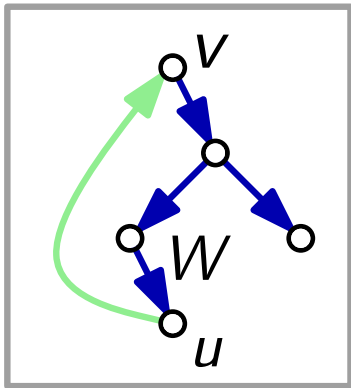
Angenommen DFS( $G$ ) liefert R-Kante  $(u, v)$ .  
 Dann ist  $u$  Nachfolger von  $v$  im DFS-Wald.

„ $\Leftarrow$ “

# Kreisfrei $\Leftrightarrow$ keine R-Kanten

**Lem.** Ein gerichteter Graph  $G$  ist kreisfrei  
 $\Leftrightarrow$  DFS( $G$ ) liefert keine Rückwärtskanten.

**Beweis.** „ $\Rightarrow$ “ Sei  $G$  kreisfrei.



Angenommen DFS( $G$ ) liefert R-Kante  $(u, v)$ .

Dann ist  $u$  Nachfolger von  $v$  im DFS-Wald.

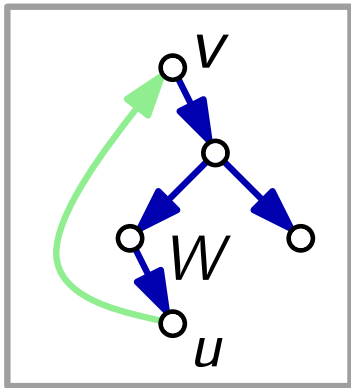
D.h.  $G$  enthält einen gerichteten  $v$ - $u$ -Weg  $W$ .

„ $\Leftarrow$ “

# Kreisfrei $\Leftrightarrow$ keine R-Kanten

**Lem.** Ein gerichteter Graph  $G$  ist kreisfrei  
 $\Leftrightarrow$  DFS( $G$ ) liefert keine Rückwärtskanten.

**Beweis.** „ $\Rightarrow$ “ Sei  $G$  kreisfrei.



Angenommen DFS( $G$ ) liefert R-Kante  $(u, v)$ .  
 Dann ist  $u$  Nachfolger von  $v$  im DFS-Wald.  
 D.h.  $G$  enthält einen gerichteten  $v$ - $u$ -Weg  $W$ .  
 Aber dann ist  $W \oplus (u, v)$  ein gerichteter Kreis.

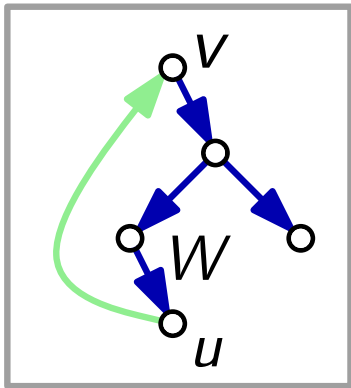
„ $\Leftarrow$ “




# Kreisfrei $\Leftrightarrow$ keine R-Kanten

**Lem.** Ein gerichteter Graph  $G$  ist kreisfrei  
 $\Leftrightarrow$  DFS( $G$ ) liefert keine Rückwärtskanten.

**Beweis.** „ $\Rightarrow$ “ Sei  $G$  kreisfrei.



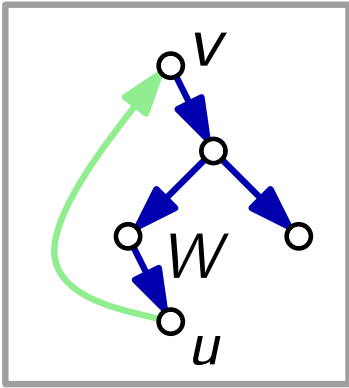
Angenommen DFS( $G$ ) liefert R-Kante  $(u, v)$ .  
 Dann ist  $u$  Nachfolger von  $v$  im DFS-Wald.  
 D.h.  $G$  enthält einen gerichteten  $v$ - $u$ -Weg  $W$ .  
 Aber dann ist  $W \oplus (u, v)$  ein gerichteter Kreis. 

„ $\Leftarrow$ “

# Kreisfrei $\Leftrightarrow$ keine R-Kanten

**Lem.** Ein gerichteter Graph  $G$  ist kreisfrei  
 $\Leftrightarrow$  DFS( $G$ ) liefert keine Rückwärtskanten.

**Beweis.** „ $\Rightarrow$ “ Sei  $G$  kreisfrei.



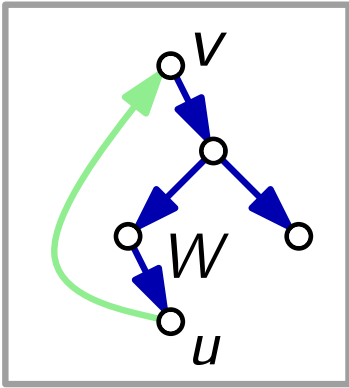
Angenommen DFS( $G$ ) liefert R-Kante  $(u, v)$ .  
 Dann ist  $u$  Nachfolger von  $v$  im DFS-Wald.  
 D.h.  $G$  enthält einen gerichteten  $v$ - $u$ -Weg  $W$ .  
 Aber dann ist  $W \oplus (u, v)$  ein gerichteter Kreis. ⚡

„ $\Leftarrow$ “ DFS( $G$ ) liefere keine R-Kanten.

# Kreisfrei $\Leftrightarrow$ keine R-Kanten

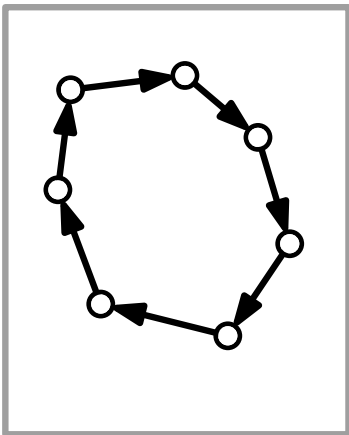
**Lem.** Ein gerichteter Graph  $G$  ist kreisfrei  
 $\Leftrightarrow$  DFS( $G$ ) liefert keine Rückwärtskanten.

**Beweis.** „ $\Rightarrow$ “ Sei  $G$  kreisfrei.



Angenommen DFS( $G$ ) liefert R-Kante  $(u, v)$ .  
 Dann ist  $u$  Nachfolger von  $v$  im DFS-Wald.  
 D.h.  $G$  enthält einen gerichteten  $v$ - $u$ -Weg  $W$ .  
 Aber dann ist  $W \oplus (u, v)$  ein gerichteter Kreis. ⚡

„ $\Leftarrow$ “ DFS( $G$ ) liefere keine R-Kanten.

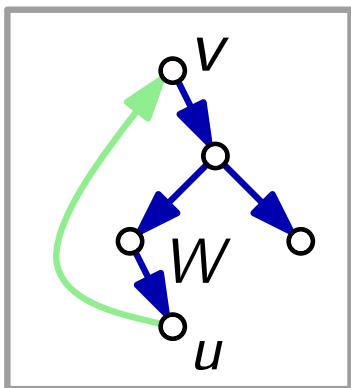


Ang.  $G$  enthält trotzdem Kreis  $C = \langle v_1, \dots, v_k \rangle$ .

# Kreisfrei $\Leftrightarrow$ keine R-Kanten

**Lem.** Ein gerichteter Graph  $G$  ist kreisfrei  
 $\Leftrightarrow$  DFS( $G$ ) liefert keine Rückwärtskanten.


**Beweis.** „ $\Rightarrow$ “ Sei  $G$  kreisfrei.



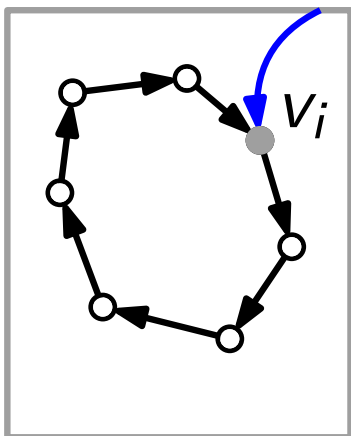
Angenommen DFS( $G$ ) liefert R-Kante  $(u, v)$ .

Dann ist  $u$  Nachfolger von  $v$  im DFS-Wald.

D.h.  $G$  enthält einen gerichteten  $v$ - $u$ -Weg  $W$ .

Aber dann ist  $W \oplus (u, v)$  ein gerichteter Kreis. 

„ $\Leftarrow$ “ DFS( $G$ ) liefere keine R-Kanten.



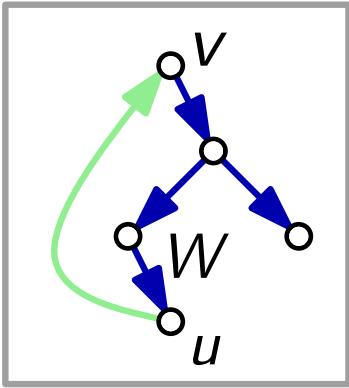
Ang.  $G$  enthält trotzdem Kreis  $C = \langle v_1, \dots, v_k \rangle$ .

Sei  $v_i$  der 1. Knoten in  $C$ , den DFS( $G$ ) erreicht.

# Kreisfrei $\Leftrightarrow$ keine R-Kanten

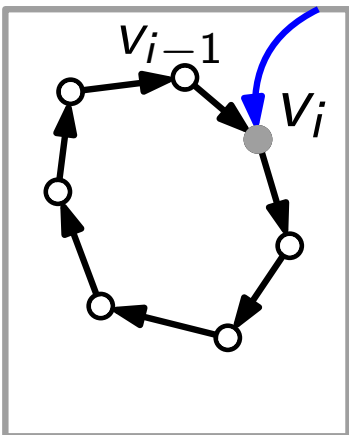
**Lem.** Ein gerichteter Graph  $G$  ist kreisfrei  
 $\Leftrightarrow$  DFS( $G$ ) liefert keine Rückwärtskanten.

**Beweis.** „ $\Rightarrow$ “ Sei  $G$  kreisfrei.



Angenommen DFS( $G$ ) liefert R-Kante  $(u, v)$ .  
 Dann ist  $u$  Nachfolger von  $v$  im DFS-Wald.  
 D.h.  $G$  enthält einen gerichteten  $v$ - $u$ -Weg  $W$ .  
 Aber dann ist  $W \oplus (u, v)$  ein gerichteter Kreis. ⚡

„ $\Leftarrow$ “ DFS( $G$ ) liefere keine R-Kanten.

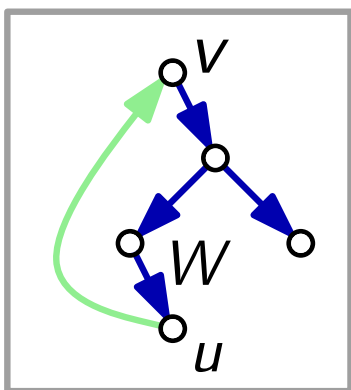


Ang.  $G$  enthält trotzdem Kreis  $C = \langle v_1, \dots, v_k \rangle$ .  
 Sei  $v_i$  der 1. Knoten in  $C$ , den DFS( $G$ ) erreicht.  
 Es gibt einen Weg von  $v_i$  nach  $v_{i-1}$  in  $G$ .

# Kreisfrei $\Leftrightarrow$ keine R-Kanten

**Lem.** Ein gerichteter Graph  $G$  ist kreisfrei  
 $\Leftrightarrow$  DFS( $G$ ) liefert keine Rückwärtskanten.


**Beweis.** „ $\Rightarrow$ “ Sei  $G$  kreisfrei.



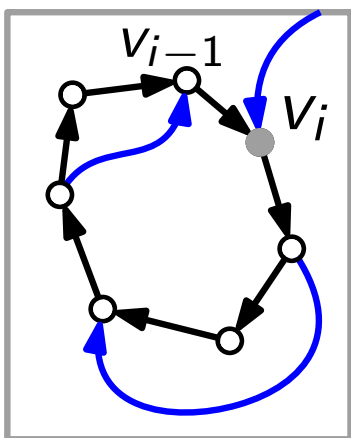
Angenommen DFS( $G$ ) liefert R-Kante  $(u, v)$ .

Dann ist  $u$  Nachfolger von  $v$  im DFS-Wald.

D.h.  $G$  enthält einen gerichteten  $v$ - $u$ -Weg  $W$ .

Aber dann ist  $W \oplus (u, v)$  ein gerichteter Kreis. 

„ $\Leftarrow$ “ DFS( $G$ ) liefere keine R-Kanten.



Ang.  $G$  enthält trotzdem Kreis  $C = \langle v_1, \dots, v_k \rangle$ .

Sei  $v_i$  der 1. Knoten in  $C$ , den DFS( $G$ ) erreicht.

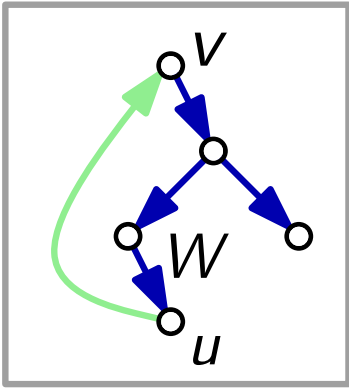
Es gibt einen Weg von  $v_i$  nach  $v_{i-1}$  in  $G$ .

$\Rightarrow$  DFS gelangt zu  $v_{i-1}$ , solange  $v_i$  grau ist.

# Kreisfrei $\Leftrightarrow$ keine R-Kanten

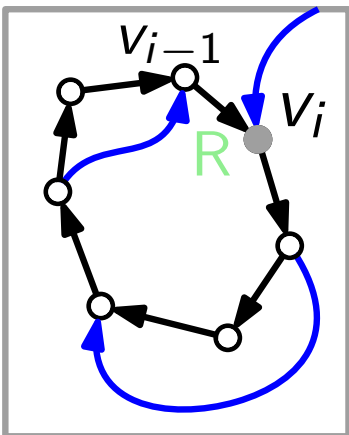
**Lem.** Ein gerichteter Graph  $G$  ist kreisfrei  
 $\Leftrightarrow$  DFS( $G$ ) liefert keine Rückwärtskanten.

**Beweis.** „ $\Rightarrow$ “ Sei  $G$  kreisfrei.



Angenommen DFS( $G$ ) liefert R-Kante  $(u, v)$ .  
 Dann ist  $u$  Nachfolger von  $v$  im DFS-Wald.  
 D.h.  $G$  enthält einen gerichteten  $v$ - $u$ -Weg  $W$ .  
 Aber dann ist  $W \oplus (u, v)$  ein gerichteter Kreis. ⚡

„ $\Leftarrow$ “ DFS( $G$ ) liefere keine R-Kanten.

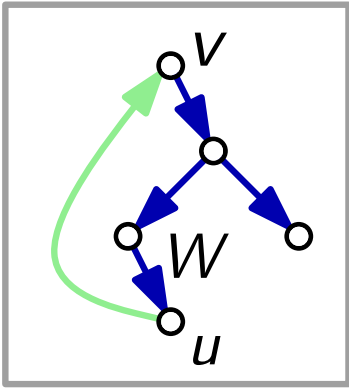


Ang.  $G$  enthält trotzdem Kreis  $C = \langle v_1, \dots, v_k \rangle$ .  
 Sei  $v_i$  der 1. Knoten in  $C$ , den DFS( $G$ ) erreicht.  
 Es gibt einen Weg von  $v_i$  nach  $v_{i-1}$  in  $G$ .  
 $\Rightarrow$  DFS gelangt zu  $v_{i-1}$ , solange  $v_i$  grau ist.  
 $\Rightarrow (v_{i-1}, v_i)$  ist R-Kante.

# Kreisfrei $\Leftrightarrow$ keine R-Kanten

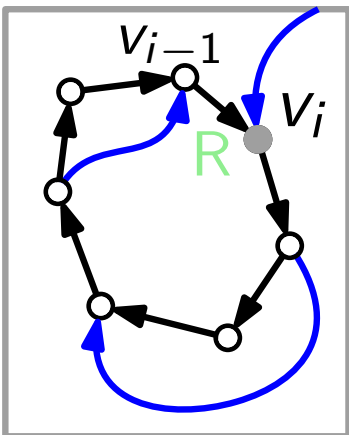
**Lem.** Ein gerichteter Graph  $G$  ist kreisfrei  
 $\Leftrightarrow$  DFS( $G$ ) liefert keine Rückwärtskanten.

**Beweis.** „ $\Rightarrow$ “ Sei  $G$  kreisfrei.



Angenommen DFS( $G$ ) liefert R-Kante  $(u, v)$ .  
 Dann ist  $u$  Nachfolger von  $v$  im DFS-Wald.  
 D.h.  $G$  enthält einen gerichteten  $v$ - $u$ -Weg  $W$ .  
 Aber dann ist  $W \oplus (u, v)$  ein gerichteter Kreis. ⚡

„ $\Leftarrow$ “ DFS( $G$ ) liefere keine R-Kanten.



Ang.  $G$  enthält trotzdem Kreis  $C = \langle v_1, \dots, v_k \rangle$ .  
 Sei  $v_i$  der 1. Knoten in  $C$ , den DFS( $G$ ) erreicht.  
 Es gibt einen Weg von  $v_i$  nach  $v_{i-1}$  in  $G$ .  
 $\Rightarrow$  DFS gelangt zu  $v_{i-1}$ , solange  $v_i$  grau ist.  
 $\Rightarrow (v_{i-1}, v_i)$  ist R-Kante. ⚡ □



# Korrektheit von TopologicalSort

**Satz.** Sei  $G$  ein gerichteter kreisfreier Graph. Dann liefert  $\text{TopologicalSort}(G)$  eine topologische Sortierung von  $G$ .

# Korrektheit von TopologicalSort

**Satz.** Sei  $G$  ein gerichteter kreisfreier Graph. Dann liefert  $\text{TopologicalSort}(G)$  eine topologische Sortierung von  $G$ .

*Beweis.* Sei  $L = \langle v_n, v_{n-1}, \dots, v_1 \rangle = \text{TopologicalSort}(G)$ .

# Korrektheit von TopologicalSort

**Satz.** Sei  $G$  ein gerichteter kreisfreier Graph. Dann liefert  $\text{TopologicalSort}(G)$  eine topologische Sortierung von  $G$ .

*Beweis.* Sei  $L = \langle v_n, v_{n-1}, \dots, v_1 \rangle = \text{TopologicalSort}(G)$ .  
Dann gilt  $v_n.f \quad \dots \quad v_1.f$ .

# Korrektheit von TopologicalSort

**Satz.** Sei  $G$  ein gerichteter kreisfreier Graph. Dann liefert  $\text{TopologicalSort}(G)$  eine topologische Sortierung von  $G$ .

*Beweis.* Sei  $L = \langle v_n, v_{n-1}, \dots, v_1 \rangle = \text{TopologicalSort}(G)$ .  
Dann gilt  $v_n.f > \dots > v_2.f > v_1.f$ .

# Korrektheit von TopologicalSort

**Satz.** Sei  $G$  ein gerichteter kreisfreier Graph. Dann liefert  $\text{TopologicalSort}(G)$  eine topologische Sortierung von  $G$ .

*Beweis.* Sei  $L = \langle v_n, v_{n-1}, \dots, v_1 \rangle = \text{TopologicalSort}(G)$ .  
Dann gilt  $v_n.f > \dots > v_2.f > v_1.f$ .

# Korrektheit von TopologicalSort

**Satz.** Sei  $G$  ein gerichteter kreisfreier Graph. Dann liefert  $\text{TopologicalSort}(G)$  eine topologische Sortierung von  $G$ .

*Beweis.* Sei  $L = \langle v_n, v_{n-1}, \dots, v_1 \rangle = \text{TopologicalSort}(G)$ .

Dann gilt  $v_n.f > \dots > v_2.f > v_1.f$ .

Sei  $(v_i, v_j)$  Kante von  $G$ .

# Korrektheit von TopologicalSort

**Satz.** Sei  $G$  ein gerichteter kreisfreier Graph. Dann liefert  $\text{TopologicalSort}(G)$  eine topologische Sortierung von  $G$ .

*Beweis.* Sei  $L = \langle v_n, v_{n-1}, \dots, v_1 \rangle = \text{TopologicalSort}(G)$ .

Dann gilt  $v_n.f > \dots > v_2.f > v_1.f$ .

Sei  $(v_i, v_j)$  Kante von  $G$ . Zu zeigen:

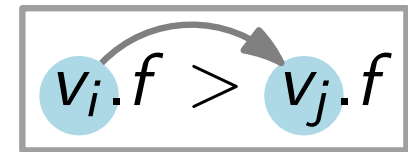
# Korrektheit von TopologicalSort

**Satz.** Sei  $G$  ein gerichteter kreisfreier Graph. Dann liefert  $\text{TopologicalSort}(G)$  eine topologische Sortierung von  $G$ .

*Beweis.* Sei  $L = \langle v_n, v_{n-1}, \dots, v_1 \rangle = \text{TopologicalSort}(G)$ .

Dann gilt  $v_n.f > \dots > v_2.f > v_1.f$ .

Sei  $(v_i, v_j)$  Kante von  $G$ . Zu zeigen:





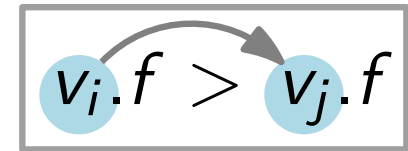
# Korrektheit von TopologicalSort

**Satz.** Sei  $G$  ein gerichteter kreisfreier Graph. Dann liefert  $\text{TopologicalSort}(G)$  eine topologische Sortierung von  $G$ .

*Beweis.* Sei  $L = \langle v_n, v_{n-1}, \dots, v_1 \rangle = \text{TopologicalSort}(G)$ .

Dann gilt  $v_n.f > \dots > v_2.f > v_1.f$ .

Sei  $(v_i, v_j)$  Kante von  $G$ . Zu zeigen:



Welche Farbe hat  $v_j$ , wenn DFS  $(v_i, v_j)$  überschreitet?

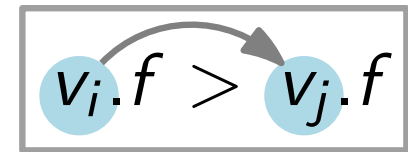
# Korrektheit von TopologicalSort

**Satz.** Sei  $G$  ein gerichteter kreisfreier Graph. Dann liefert  $\text{TopologicalSort}(G)$  eine topologische Sortierung von  $G$ .

**Beweis.** Sei  $L = \langle v_n, v_{n-1}, \dots, v_1 \rangle = \text{TopologicalSort}(G)$ .

Dann gilt  $v_n.f > \dots > v_2.f > v_1.f$ .

Sei  $(v_i, v_j)$  Kante von  $G$ . Zu zeigen:



Welche Farbe hat  $v_j$ , wenn DFS  $(v_i, v_j)$  überschreitet?



–  $v_j$  grau



–  $v_j$  weiß



–  $v_j$  schwarz

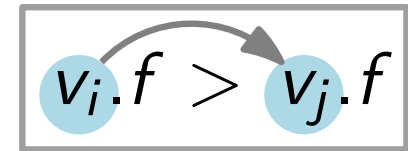
# Korrektheit von TopologicalSort

**Satz.** Sei  $G$  ein gerichteter kreisfreier Graph. Dann liefert  $\text{TopologicalSort}(G)$  eine topologische Sortierung von  $G$ .

**Beweis.** Sei  $L = \langle v_n, v_{n-1}, \dots, v_1 \rangle = \text{TopologicalSort}(G)$ .

Dann gilt  $v_n.f > \dots > v_2.f > v_1.f$ .

Sei  $(v_i, v_j)$  Kante von  $G$ . Zu zeigen:



Welche Farbe hat  $v_j$ , wenn DFS  $(v_i, v_j)$  überschreitet?



–  $v_j$  grau  $\Rightarrow$



–  $v_j$  weiß



–  $v_j$  schwarz

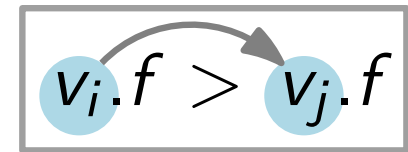
# Korrektheit von TopologicalSort

**Satz.** Sei  $G$  ein gerichteter kreisfreier Graph. Dann liefert  $\text{TopologicalSort}(G)$  eine topologische Sortierung von  $G$ .

**Beweis.** Sei  $L = \langle v_n, v_{n-1}, \dots, v_1 \rangle = \text{TopologicalSort}(G)$ .

Dann gilt  $v_n.f > \dots > v_2.f > v_1.f$ .

Sei  $(v_i, v_j)$  Kante von  $G$ . Zu zeigen:



Welche Farbe hat  $v_j$ , wenn DFS  $(v_i, v_j)$  überschreitet?



–  $v_j$  grau  $\Rightarrow (v_i, v_j)$  ist R-Kante



–  $v_j$  weiß



–  $v_j$  schwarz

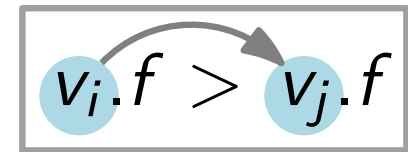
# Korrektheit von TopologicalSort

**Satz.** Sei  $G$  ein gerichteter kreisfreier Graph. Dann liefert  $\text{TopologicalSort}(G)$  eine topologische Sortierung von  $G$ .

**Beweis.** Sei  $L = \langle v_n, v_{n-1}, \dots, v_1 \rangle = \text{TopologicalSort}(G)$ .

Dann gilt  $v_n.f > \dots > v_2.f > v_1.f$ .

Sei  $(v_i, v_j)$  Kante von  $G$ . Zu zeigen:



Welche Farbe hat  $v_j$ , wenn DFS  $(v_i, v_j)$  überschreitet?



–  $v_j$  grau

$\Rightarrow (v_i, v_j)$  ist R-Kante 



–  $v_j$  weiß



–  $v_j$  schwarz

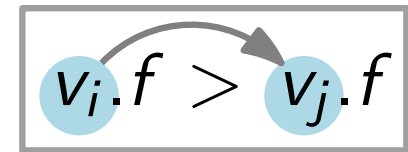
# Korrektheit von TopologicalSort

**Satz.** Sei  $G$  ein gerichteter kreisfreier Graph. Dann liefert  $\text{TopologicalSort}(G)$  eine topologische Sortierung von  $G$ .

**Beweis.** Sei  $L = \langle v_n, v_{n-1}, \dots, v_1 \rangle = \text{TopologicalSort}(G)$ .

Dann gilt  $v_n.f > \dots > v_2.f > v_1.f$ .

Sei  $(v_i, v_j)$  Kante von  $G$ . Zu zeigen:



Welche Farbe hat  $v_j$ , wenn DFS  $(v_i, v_j)$  überschreitet?



–  $v_j$  grau

$\Rightarrow (v_i, v_j)$  ist R-Kante



Widerspruch zu  
Lemma:  $G$  kreisfrei!



–  $v_j$  weiß



–  $v_j$  schwarz

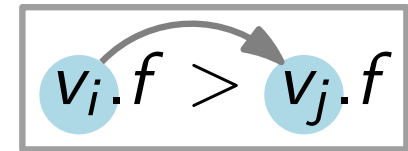
# Korrektheit von TopologicalSort

**Satz.** Sei  $G$  ein gerichteter kreisfreier Graph. Dann liefert  $\text{TopologicalSort}(G)$  eine topologische Sortierung von  $G$ .

**Beweis.** Sei  $L = \langle v_n, v_{n-1}, \dots, v_1 \rangle = \text{TopologicalSort}(G)$ .

Dann gilt  $v_n.f > \dots > v_2.f > v_1.f$ .

Sei  $(v_i, v_j)$  Kante von  $G$ . Zu zeigen:



Welche Farbe hat  $v_j$ , wenn DFS  $(v_i, v_j)$  überschreitet?



–  $v_j$  grau

$\Rightarrow (v_i, v_j)$  ist R-Kante



Widerspruch zu  
Lemma:  $G$  kreisfrei!



–  $v_j$  weiß

$\Rightarrow$



–  $v_j$  schwarz

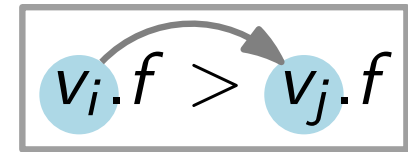
# Korrektheit von TopologicalSort

**Satz.** Sei  $G$  ein gerichteter kreisfreier Graph. Dann liefert  $\text{TopologicalSort}(G)$  eine topologische Sortierung von  $G$ .

**Beweis.** Sei  $L = \langle v_n, v_{n-1}, \dots, v_1 \rangle = \text{TopologicalSort}(G)$ .

Dann gilt  $v_n.f > \dots > v_2.f > v_1.f$ .

Sei  $(v_i, v_j)$  Kante von  $G$ . Zu zeigen:



Welche Farbe hat  $v_j$ , wenn DFS  $(v_i, v_j)$  überschreitet?



–  $v_j$  grau

$\Rightarrow (v_i, v_j)$  ist R-Kante



Widerspruch zu  
Lemma:  $G$  kreisfrei!



–  $v_j$  weiß

$\Rightarrow v_j$  Nachfolger von  $v_i$



–  $v_j$  schwarz



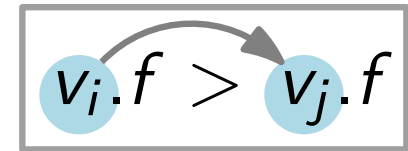
# Korrektheit von TopologicalSort

**Satz.** Sei  $G$  ein gerichteter kreisfreier Graph. Dann liefert  $\text{TopologicalSort}(G)$  eine topologische Sortierung von  $G$ .

**Beweis.** Sei  $L = \langle v_n, v_{n-1}, \dots, v_1 \rangle = \text{TopologicalSort}(G)$ .

Dann gilt  $v_n.f > \dots > v_2.f > v_1.f$ .

Sei  $(v_i, v_j)$  Kante von  $G$ . Zu zeigen:



Welche Farbe hat  $v_j$ , wenn DFS  $(v_i, v_j)$  überschreitet?



–  $v_j$  grau

$\Rightarrow (v_i, v_j)$  ist R-Kante



Widerspruch zu  
Lemma:  $G$  kreisfrei!



–  $v_j$  weiß

$\Rightarrow v_j$  Nachfolger von  $v_i \Rightarrow$



–  $v_j$  schwarz

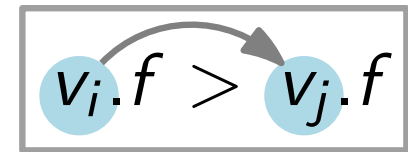
# Korrektheit von TopologicalSort

**Satz.** Sei  $G$  ein gerichteter kreisfreier Graph. Dann liefert  $\text{TopologicalSort}(G)$  eine topologische Sortierung von  $G$ .

**Beweis.** Sei  $L = \langle v_n, v_{n-1}, \dots, v_1 \rangle = \text{TopologicalSort}(G)$ .

Dann gilt  $v_n.f > \dots > v_2.f > v_1.f$ .

Sei  $(v_i, v_j)$  Kante von  $G$ . Zu zeigen:



Welche Farbe hat  $v_j$ , wenn DFS  $(v_i, v_j)$  überschreitet?



–  $v_j$  grau

$\Rightarrow (v_i, v_j)$  ist R-Kante



Widerspruch zu  
Lemma:  $G$  kreisfrei!



–  $v_j$  weiß

$\Rightarrow v_j$  Nachfolger von  $v_i \Rightarrow v_i.f > v_j.f$



–  $v_j$  schwarz

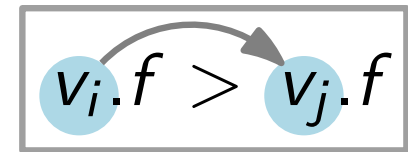
# Korrektheit von TopologicalSort

**Satz.** Sei  $G$  ein gerichteter kreisfreier Graph. Dann liefert  $\text{TopologicalSort}(G)$  eine topologische Sortierung von  $G$ .

**Beweis.** Sei  $L = \langle v_n, v_{n-1}, \dots, v_1 \rangle = \text{TopologicalSort}(G)$ .

Dann gilt  $v_n.f > \dots > v_2.f > v_1.f$ .

Sei  $(v_i, v_j)$  Kante von  $G$ . Zu zeigen:



Welche Farbe hat  $v_j$ , wenn DFS  $(v_i, v_j)$  überschreitet?



–  $v_j$  grau

$\Rightarrow (v_i, v_j)$  ist R-Kante



Widerspruch zu  
Lemma:  $G$  kreisfrei!



–  $v_j$  weiß

$\Rightarrow v_j$  Nachfolger von  $v_i \Rightarrow v_i.f > v_j.f$  ✓



–  $v_j$  schwarz

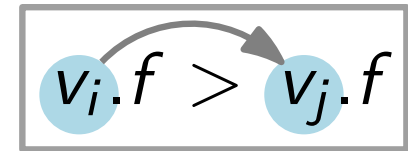
# Korrektheit von TopologicalSort

**Satz.** Sei  $G$  ein gerichteter kreisfreier Graph. Dann liefert  $\text{TopologicalSort}(G)$  eine topologische Sortierung von  $G$ .

**Beweis.** Sei  $L = \langle v_n, v_{n-1}, \dots, v_1 \rangle = \text{TopologicalSort}(G)$ .

Dann gilt  $v_n.f > \dots > v_2.f > v_1.f$ .

Sei  $(v_i, v_j)$  Kante von  $G$ . Zu zeigen:



Welche Farbe hat  $v_j$ , wenn DFS  $(v_i, v_j)$  überschreitet?



–  $v_j$  grau

$\Rightarrow (v_i, v_j)$  ist R-Kante



Widerspruch zu  
Lemma:  $G$  kreisfrei!



–  $v_j$  weiß

$\Rightarrow v_j$  Nachfolger von  $v_i \Rightarrow v_i.f > v_j.f$  ✓



–  $v_j$  schwarz  $\Rightarrow$

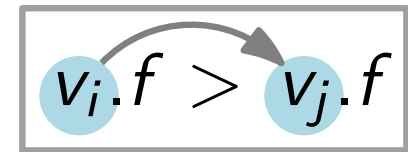
# Korrektheit von TopologicalSort

**Satz.** Sei  $G$  ein gerichteter kreisfreier Graph. Dann liefert  $\text{TopologicalSort}(G)$  eine topologische Sortierung von  $G$ .

**Beweis.** Sei  $L = \langle v_n, v_{n-1}, \dots, v_1 \rangle = \text{TopologicalSort}(G)$ .

Dann gilt  $v_n.f > \dots > v_2.f > v_1.f$ .

Sei  $(v_i, v_j)$  Kante von  $G$ . Zu zeigen:



Welche Farbe hat  $v_j$ , wenn DFS  $(v_i, v_j)$  überschreitet?



–  $v_j$  grau

$\Rightarrow (v_i, v_j)$  ist R-Kante

⚡ Widerspruch zu  
Lemma:  $G$  kreisfrei!



–  $v_j$  weiß

$\Rightarrow v_j$  Nachfolger von  $v_i \Rightarrow v_i.f > v_j.f$  ✓



–  $v_j$  schwarz

$\Rightarrow v_i.f$  noch nicht gesetzt,  $v_j.f$  gesetzt

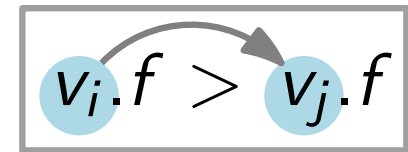
# Korrektheit von TopologicalSort

**Satz.** Sei  $G$  ein gerichteter kreisfreier Graph. Dann liefert  $\text{TopologicalSort}(G)$  eine topologische Sortierung von  $G$ .

**Beweis.** Sei  $L = \langle v_n, v_{n-1}, \dots, v_1 \rangle = \text{TopologicalSort}(G)$ .

Dann gilt  $v_n.f > \dots > v_2.f > v_1.f$ .

Sei  $(v_i, v_j)$  Kante von  $G$ . Zu zeigen:



Welche Farbe hat  $v_j$ , wenn DFS  $(v_i, v_j)$  überschreitet?



–  $v_j$  grau  $\Rightarrow (v_i, v_j)$  ist R-Kante  Widerspruch zu Lemma:  $G$  kreisfrei!



–  $v_j$  weiß  $\Rightarrow v_j$  Nachfolger von  $v_i \Rightarrow v_i.f > v_j.f$  ✓



–  $v_j$  schwarz  $\Rightarrow v_i.f$  noch nicht gesetzt,  $v_j.f$  gesetzt

$\Rightarrow$

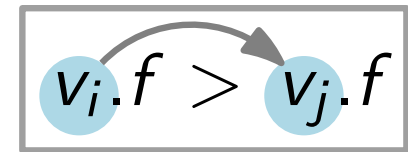
# Korrektheit von TopologicalSort

**Satz.** Sei  $G$  ein gerichteter kreisfreier Graph. Dann liefert  $\text{TopologicalSort}(G)$  eine topologische Sortierung von  $G$ .

**Beweis.** Sei  $L = \langle v_n, v_{n-1}, \dots, v_1 \rangle = \text{TopologicalSort}(G)$ .

Dann gilt  $v_n.f > \dots > v_2.f > v_1.f$ .

Sei  $(v_i, v_j)$  Kante von  $G$ . Zu zeigen:



Welche Farbe hat  $v_j$ , wenn DFS  $(v_i, v_j)$  überschreitet?



–  $v_j$  grau

$\Rightarrow (v_i, v_j)$  ist R-Kante

⚡ Widerspruch zu  
Lemma:  $G$  kreisfrei!



–  $v_j$  weiß

$\Rightarrow v_j$  Nachfolger von  $v_i \Rightarrow v_i.f > v_j.f$  ✓



–  $v_j$  schwarz

$\Rightarrow v_i.f$  noch nicht gesetzt,  $v_j.f$  gesetzt

$\Rightarrow v_i.f > v_j.f$

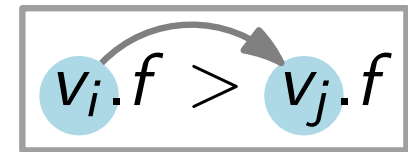
# Korrektheit von TopologicalSort

**Satz.** Sei  $G$  ein gerichteter kreisfreier Graph. Dann liefert  $\text{TopologicalSort}(G)$  eine topologische Sortierung von  $G$ .

**Beweis.** Sei  $L = \langle v_n, v_{n-1}, \dots, v_1 \rangle = \text{TopologicalSort}(G)$ .

Dann gilt  $v_n.f > \dots > v_2.f > v_1.f$ .

Sei  $(v_i, v_j)$  Kante von  $G$ . Zu zeigen:



Welche Farbe hat  $v_j$ , wenn DFS  $(v_i, v_j)$  überschreitet?



–  $v_j$  grau  $\Rightarrow (v_i, v_j)$  ist R-Kante  Widerspruch zu Lemma:  $G$  kreisfrei!



–  $v_j$  weiß  $\Rightarrow v_j$  Nachfolger von  $v_i \Rightarrow v_i.f > v_j.f$  ✓



–  $v_j$  schwarz  $\Rightarrow v_i.f$  noch nicht gesetzt,  $v_j.f$  gesetzt  
 $\Rightarrow v_i.f > v_j.f$  ✓

□



# Vergleich Durchlaufstrategien für Graphen

Breitensuche

Tiefensuche

Laufzeit

Ergebnis

Datenstruktur

Vorgehen

# Vergleich Durchlaufstrategien für Graphen

Breitensuche

Tiefensuche

---

Laufzeit

$$O(V + E)$$

Ergebnis

Datenstruktur

Vorgehen

# Vergleich Durchlaufstrategien für Graphen

Breitensuche

Tiefensuche

---

Laufzeit

$O(V + E)$

$O(V + E)$

Ergebnis

Datenstruktur

Vorgehen

# Vergleich Durchlaufstrategien für Graphen

Breitensuche

Tiefensuche

---

Laufzeit

$O(V + E)$

$O(V + E)$

Ergebnis

BFS-Baum,  
d.h. kürzeste Wege

Datenstruktur

Vorgehen

# Vergleich Durchlaufstrategien für Graphen

Breitensuche

Tiefensuche

---

Laufzeit

$O(V + E)$

$O(V + E)$

Ergebnis

BFS-Baum,  
d.h. kürzeste Wege

$d$ - und  $f$ -Werte,  
z.B. für top. Sortierung

Datenstruktur

Vorgehen

# Vergleich Durchlaufstrategien für Graphen

Breitensuche

Tiefensuche

---

Laufzeit

$O(V + E)$

$O(V + E)$

Ergebnis

BFS-Baum,  
d.h. kürzeste Wege

$d$ - und  $f$ -Werte,  
z.B. für top. Sortierung

Datenstruktur

Schlange

Vorgehen

# Vergleich Durchlaufstrategien für Graphen

Breitensuche

Tiefensuche

---

Laufzeit

$O(V + E)$

$O(V + E)$

Ergebnis

BFS-Baum,  
d.h. kürzeste Wege

$d$ - und  $f$ -Werte,  
z.B. für top. Sortierung

Datenstruktur

Schlange

Rekursion bzw. Stapel

Vorgehen

# Vergleich Durchlaufstrategien für Graphen

Breitensuche

Tiefensuche

---

Laufzeit

$O(V + E)$

$O(V + E)$

Ergebnis

BFS-Baum,  
d.h. kürzeste Wege

$d$ - und  $f$ -Werte,  
z.B. für top. Sortierung

Datenstruktur

Schlange

Rekursion bzw. Stapel

Vorgehen

nicht-lokal



# Vergleich Durchlaufstrategien für Graphen

Breitensuche

Tiefensuche

---

Laufzeit

$O(V + E)$

$O(V + E)$

Ergebnis

BFS-Baum,  
d.h. kürzeste Wege

$d$ - und  $f$ -Werte,  
z.B. für top. Sortierung

Datenstruktur

Schlange

Rekursion bzw. Stapel

Vorgehen

nicht-lokal

lokal