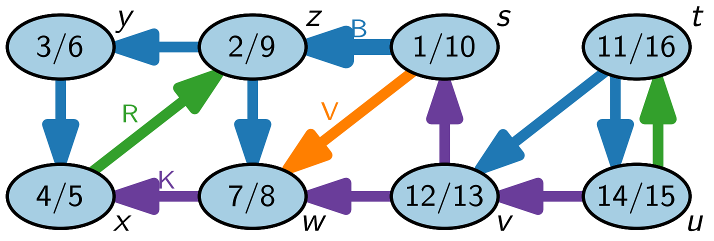
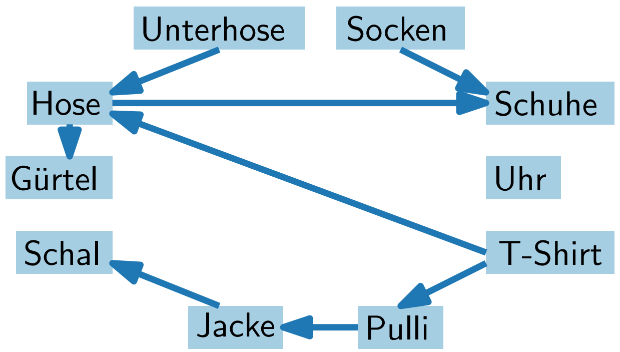
Algorithmen und Datenstrukturen

Vorlesung 20: Tiefensuche und topologische Sortierung



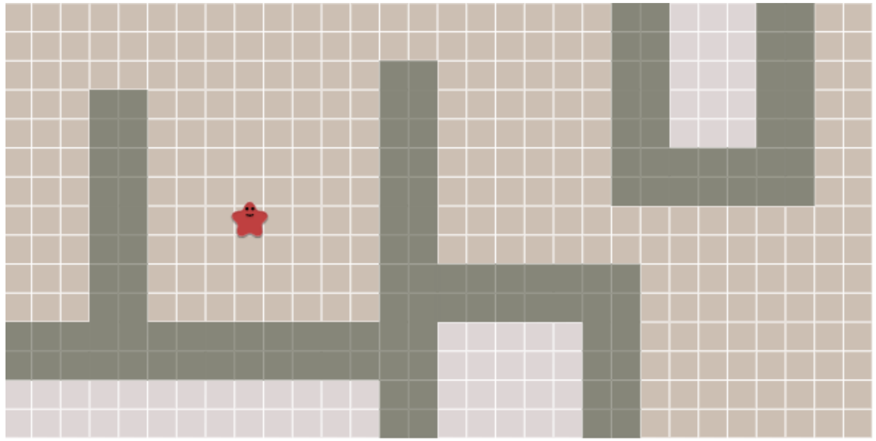


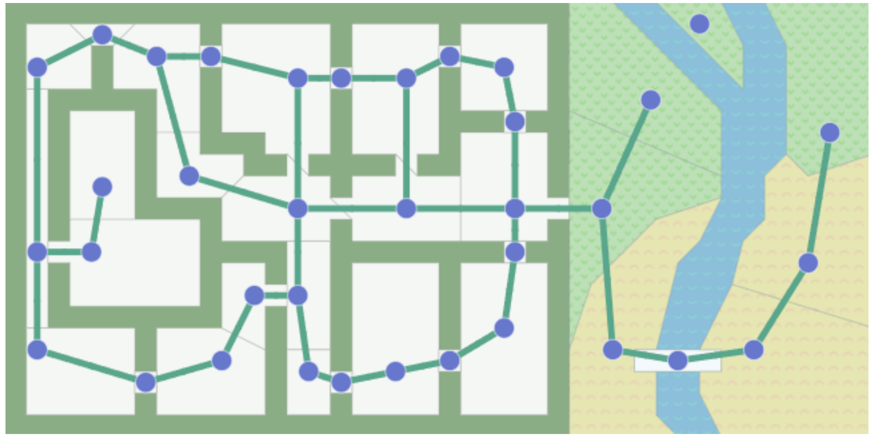
Alexander Wolff

Wintersemester 2025

Wie durchlaufe ich einen Graphen?

Wie finde ich heraus, welche Knoten von einem Startknoten s aus erreichbar sind?



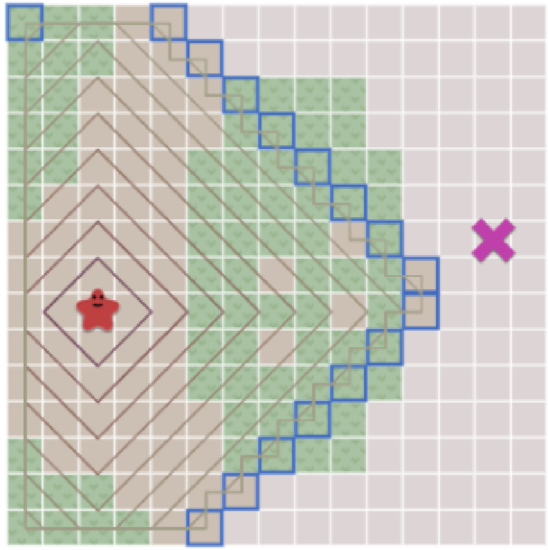


Amit Patel, "Introduction to the A^* Algorithm", Red Blob Games, 2014, https://www.redblobgames.com/pathfinding/a-star/introduction.html

1. wellenförmige Ausbreitung ab s

   Breitensuche (breadth-first search, BFS)

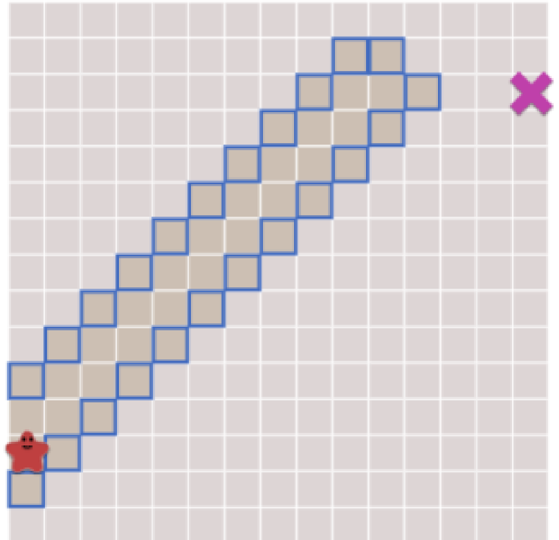
   vorletztes Mal

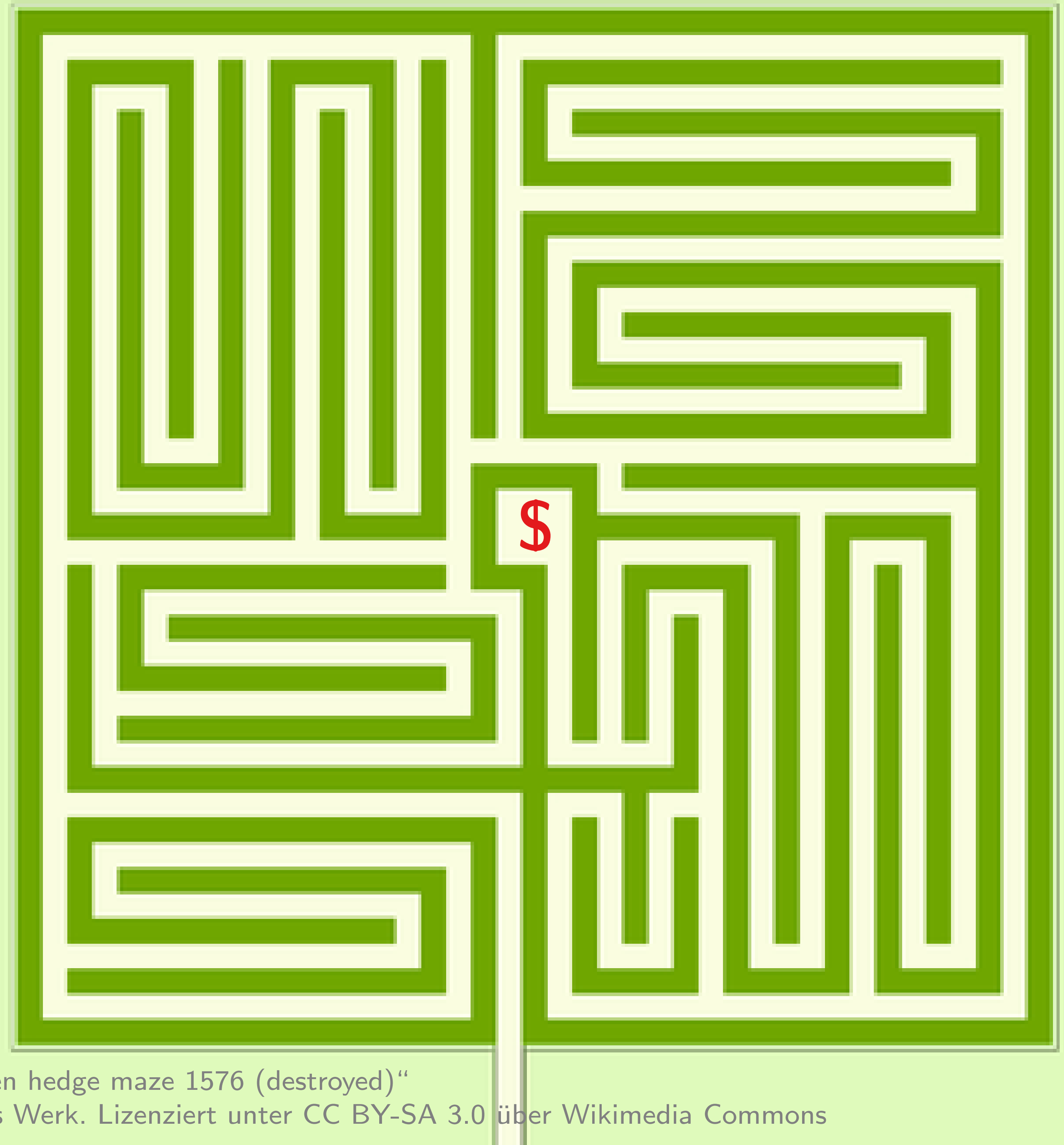


2. von s möglichst schnell weit weg

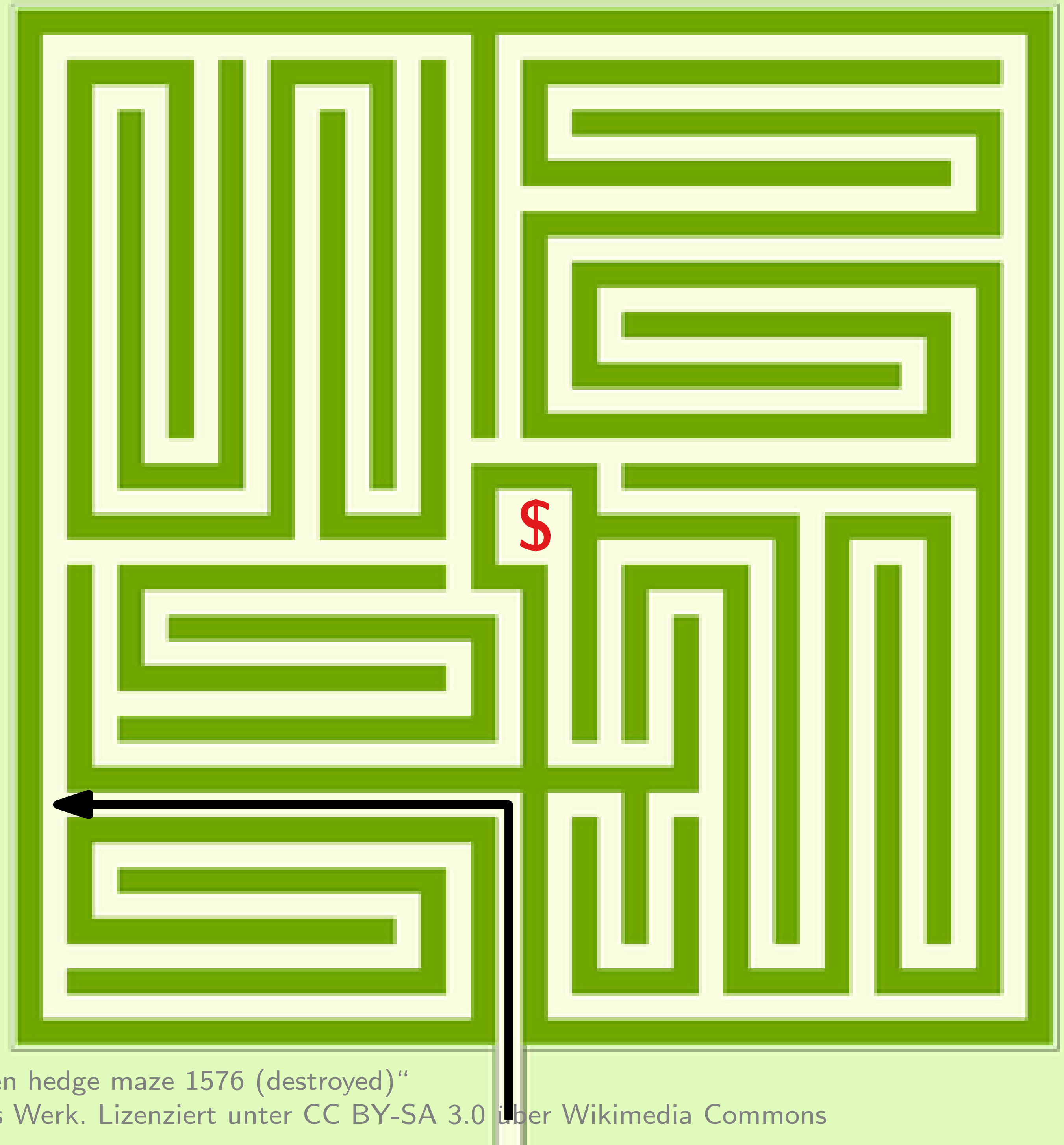
   Tiefensuche (depth-first search, DFS)

   heute

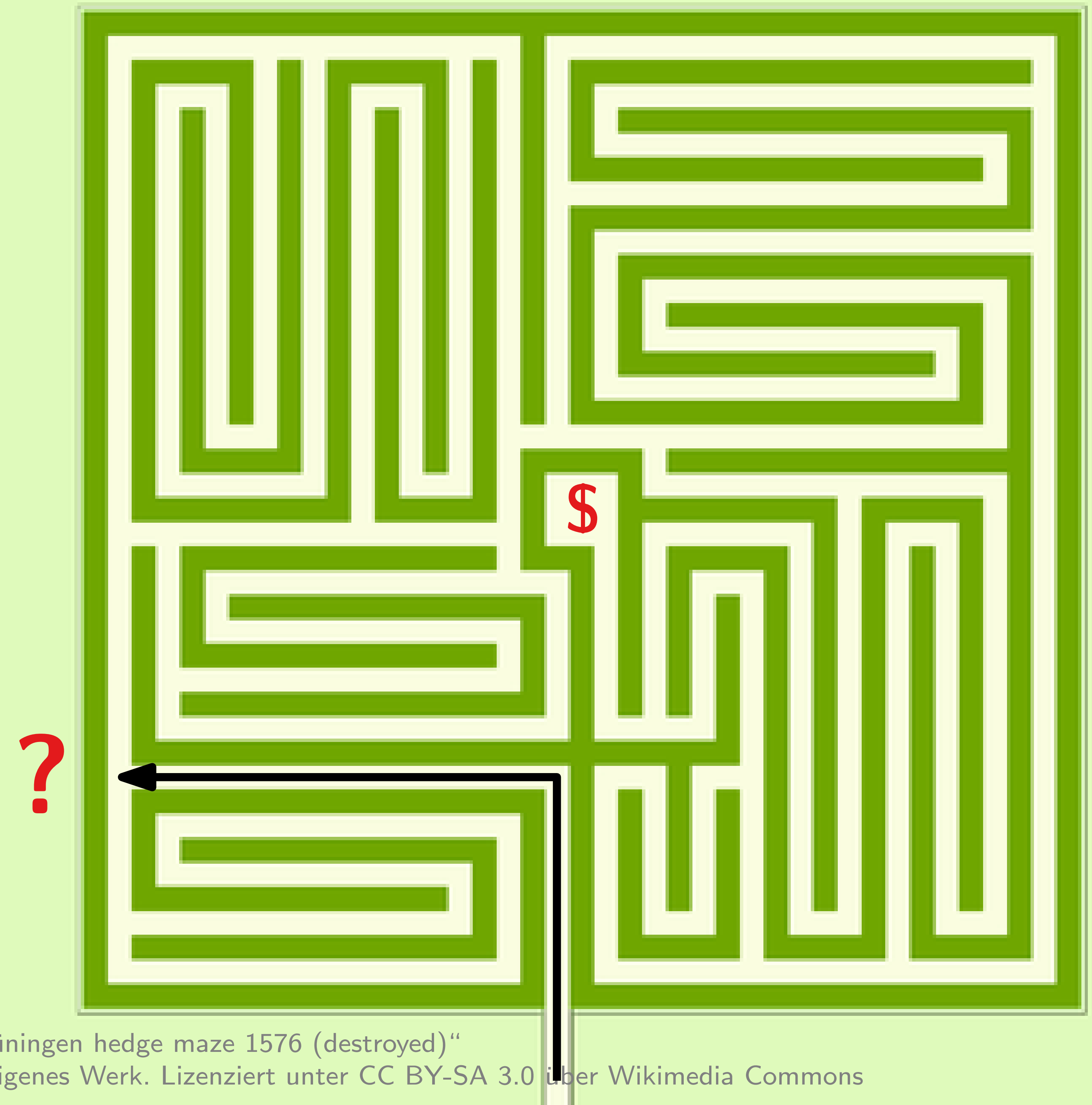


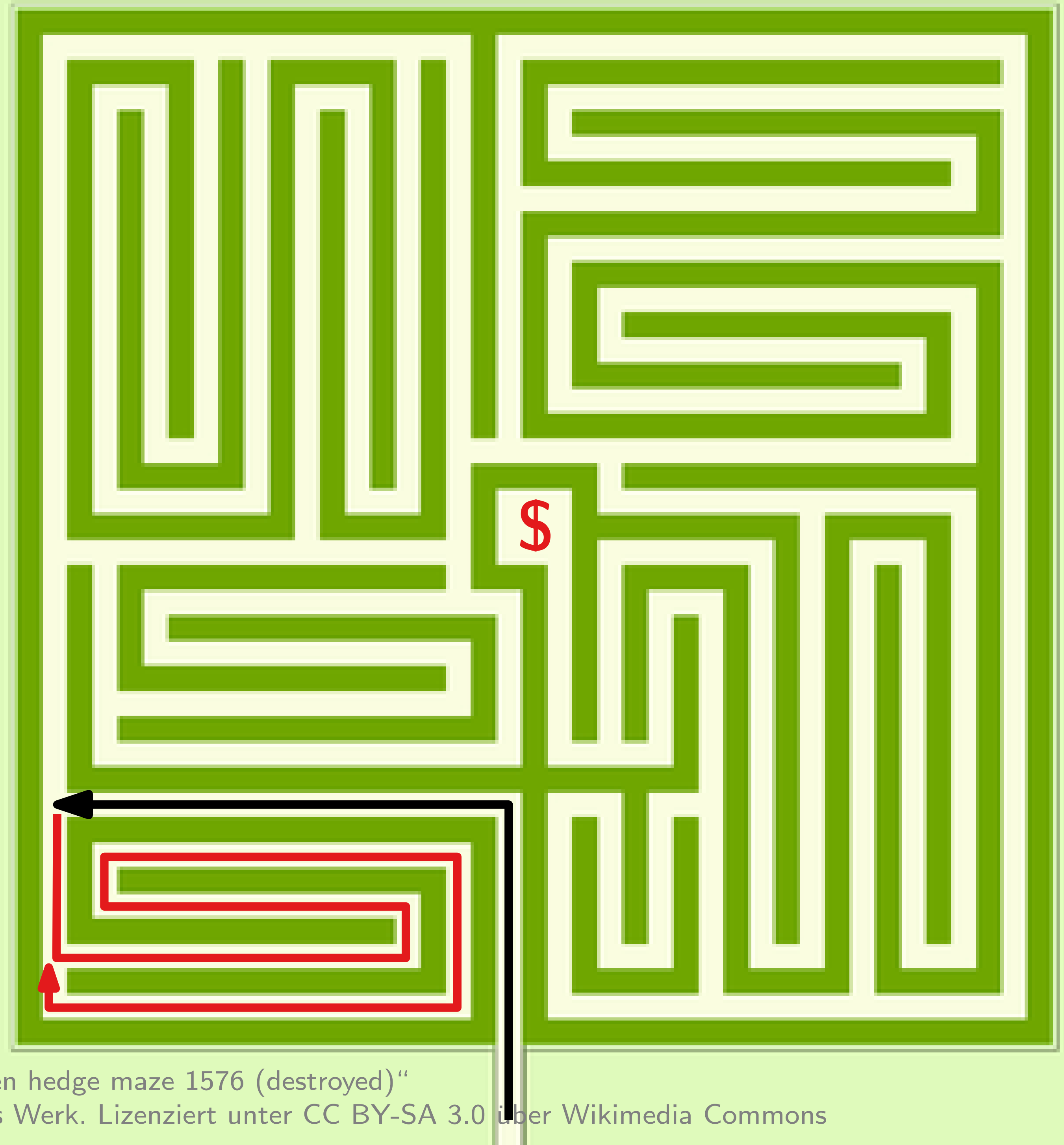


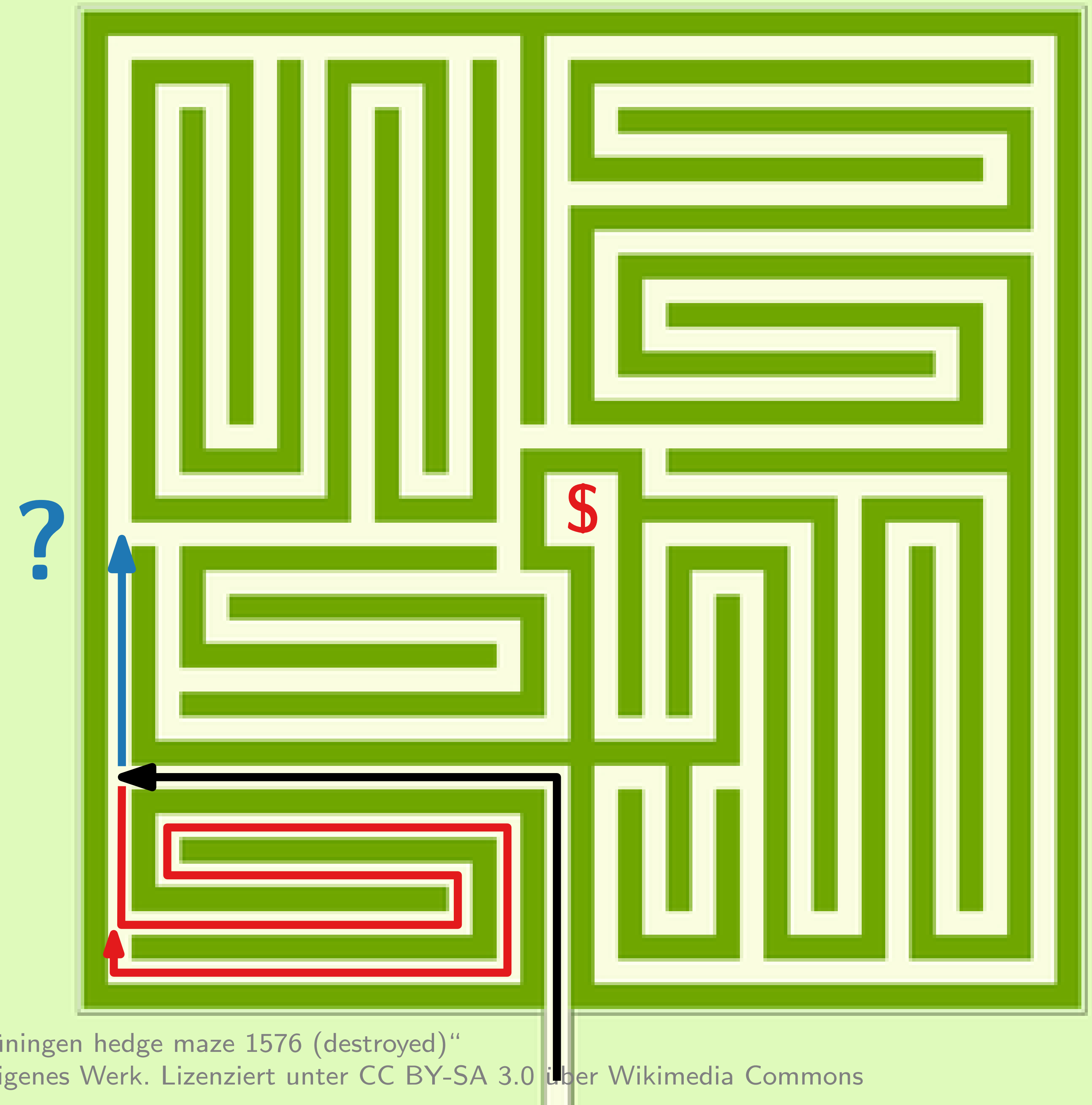
„Maze-01 Grüningen hedge maze 1576 (destroyed)“

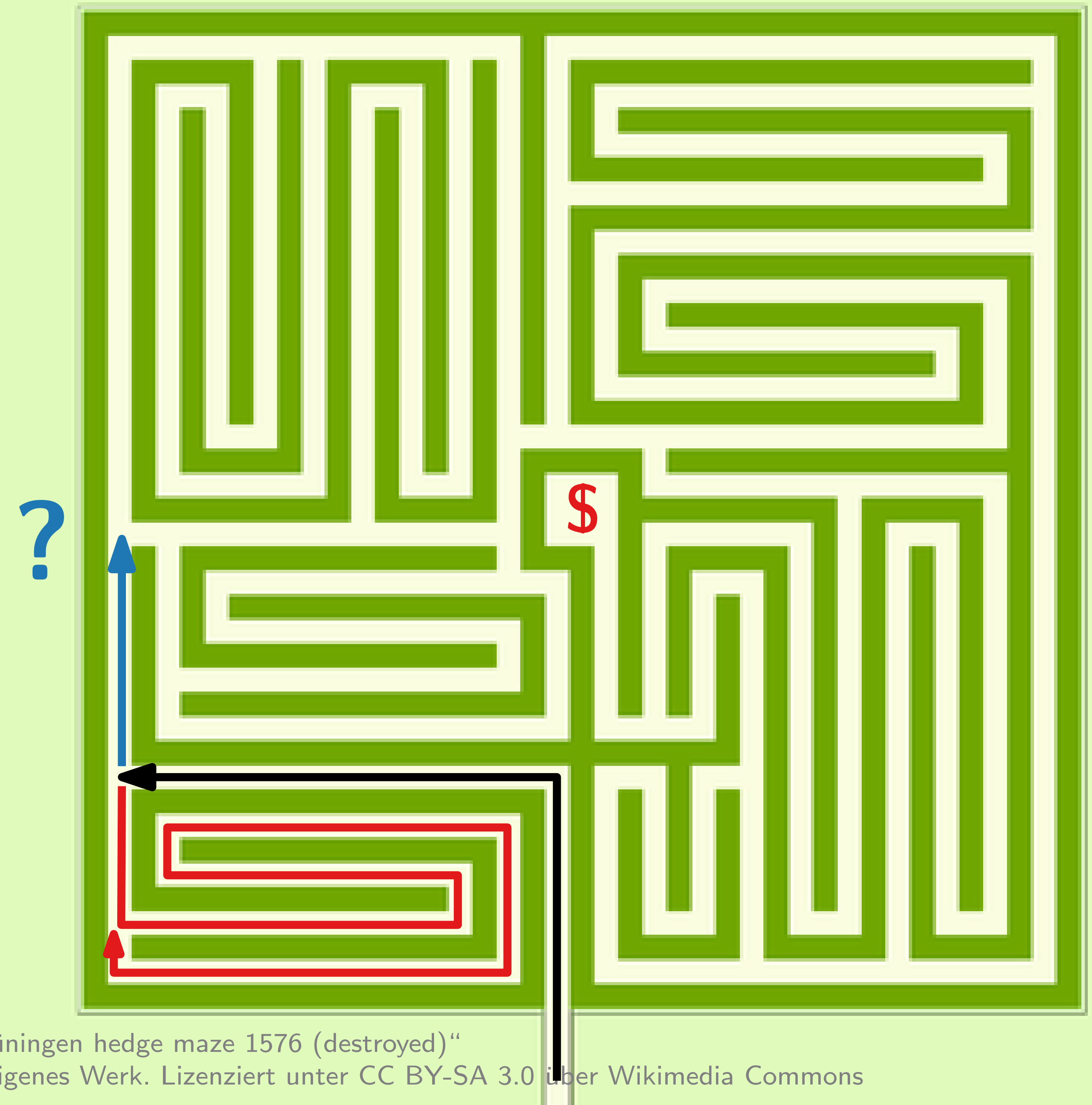


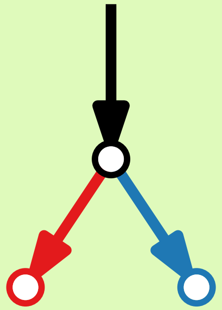
„Maze-01 Grüningen hedge maze 1576 (destroyed)“



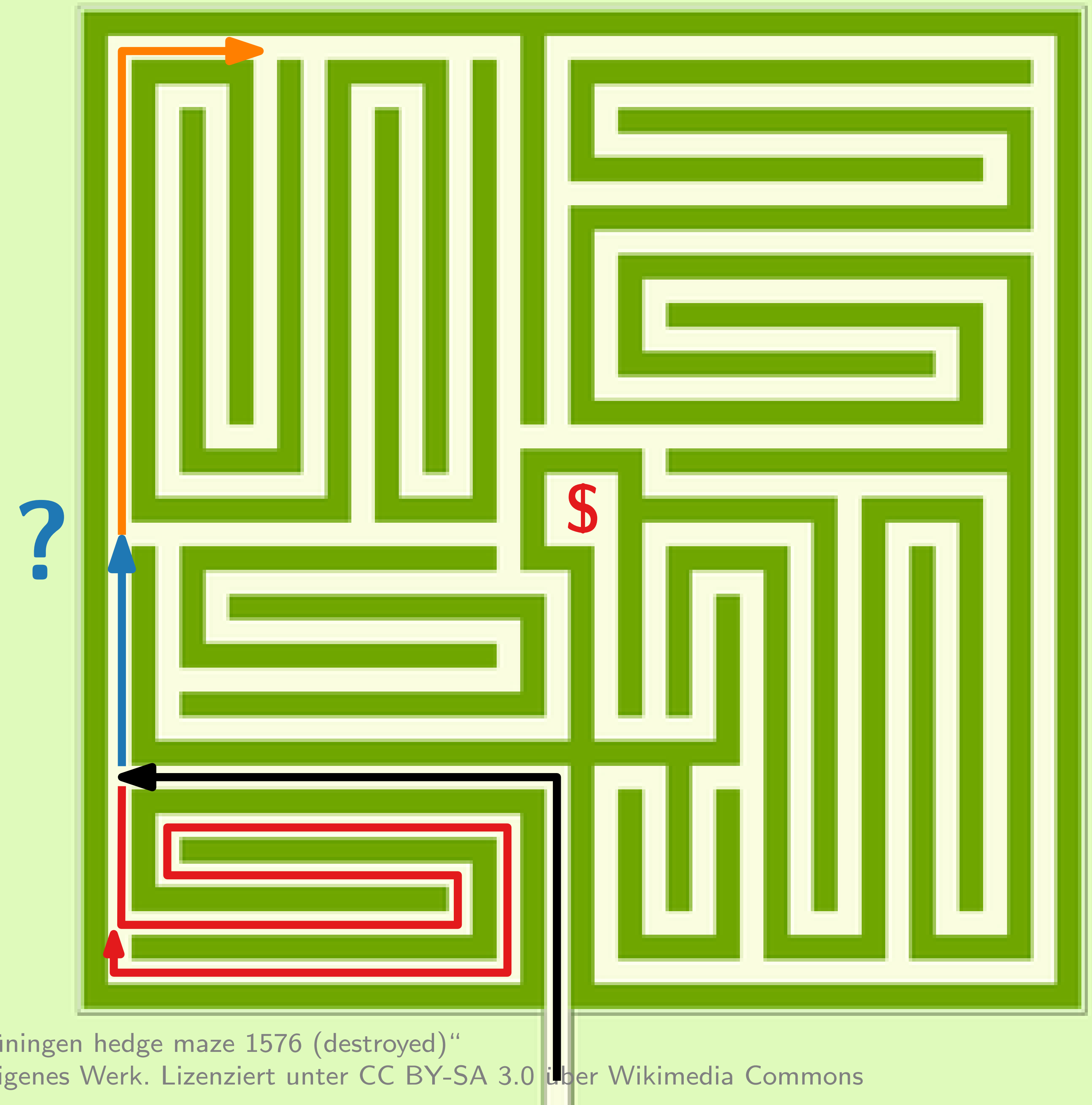


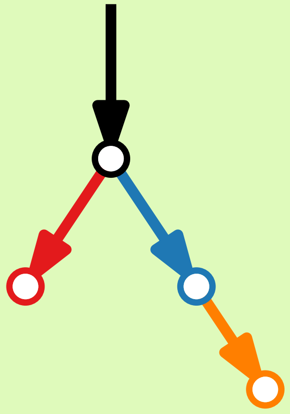


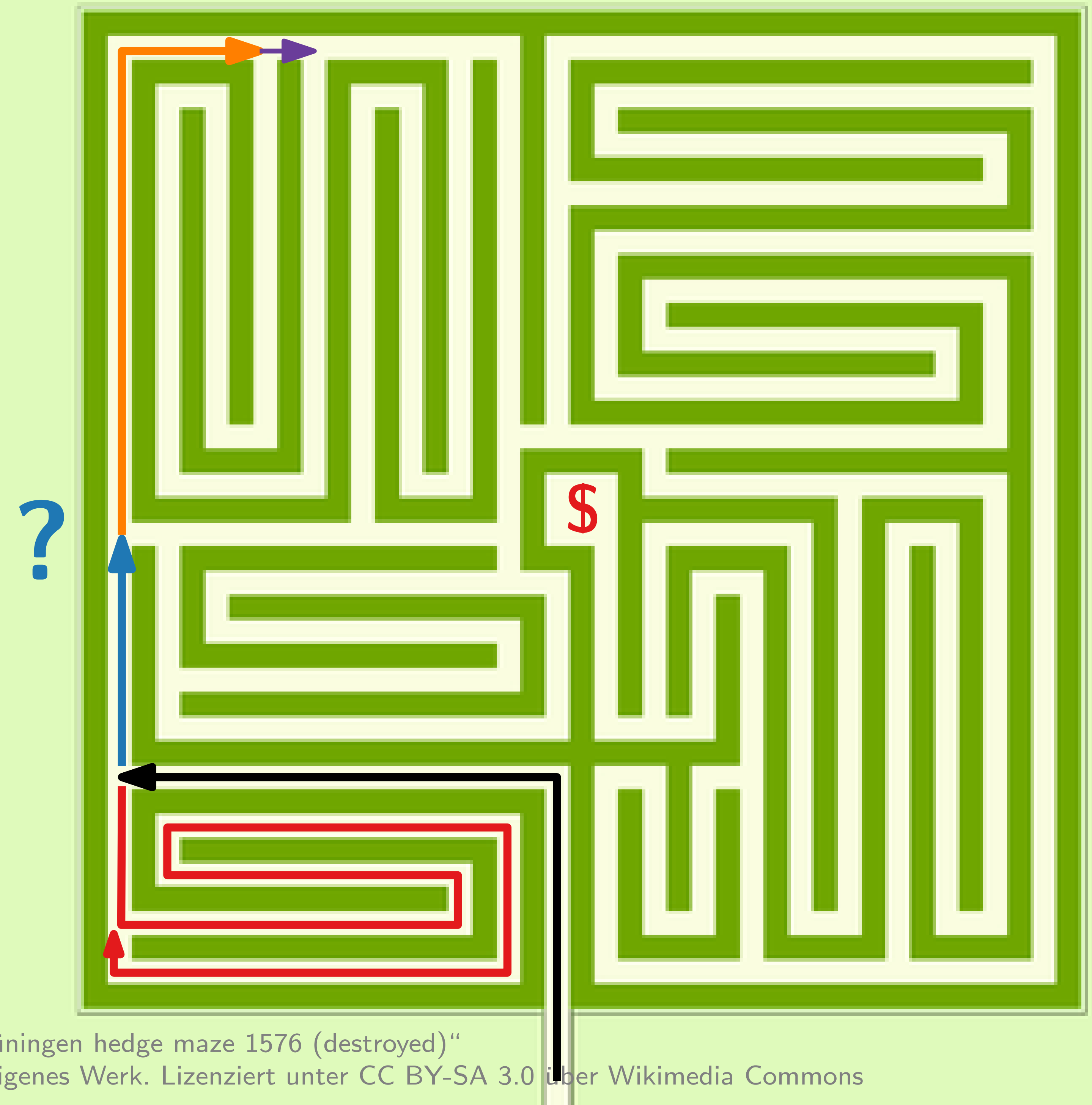


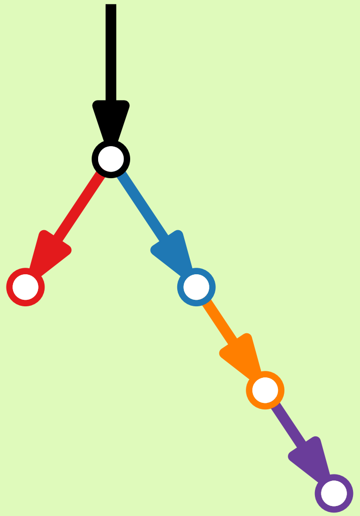


„Maze-01 Grüningen hedge maze 1576 (destroyed)“



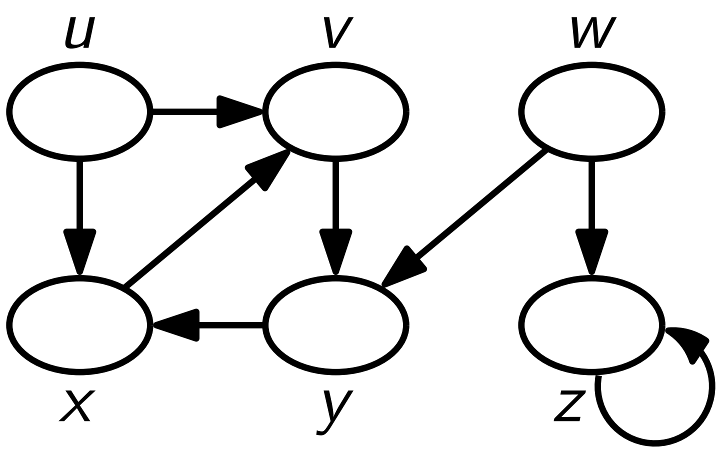






Tiefensuche

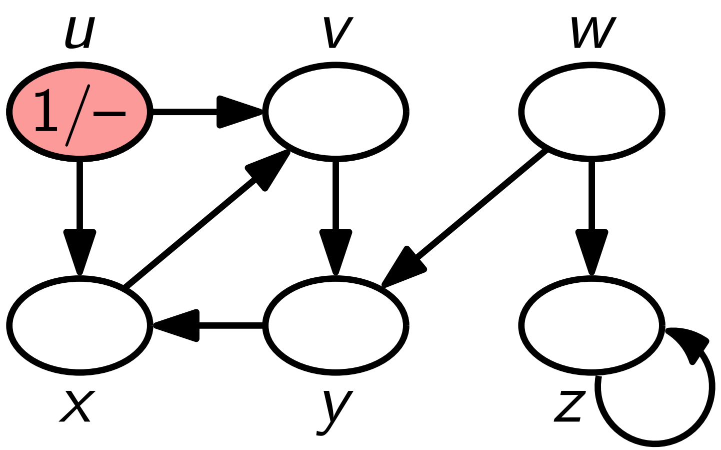
Eingabe: (un)gerichteter Graph G



Tiefensuche

Eingabe: (un)gerichteter Graph G

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

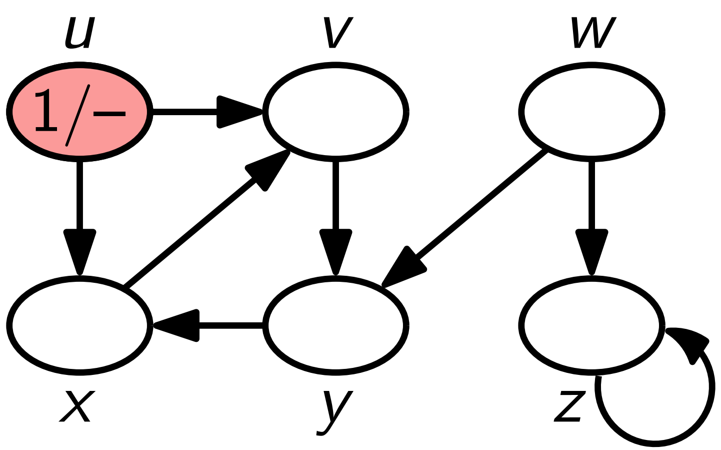


Tiefensuche

Eingabe: (un)gerichteter Graph G

discovery time

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



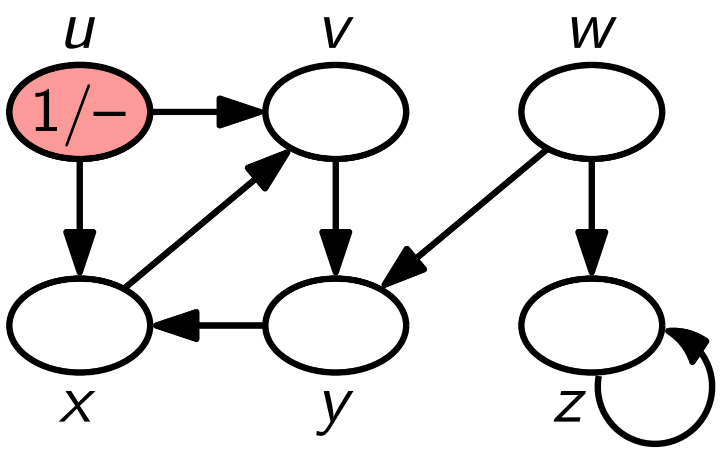
Tiefensuche

Eingabe: (un)gerichteter Graph G

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

discovery time

finish time



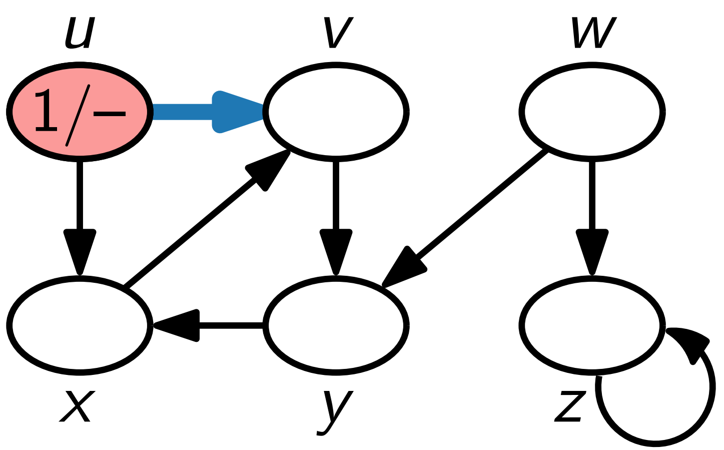
Tiefensuche

Eingabe: (un)gerichteter Graph G

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

discovery time

finish time

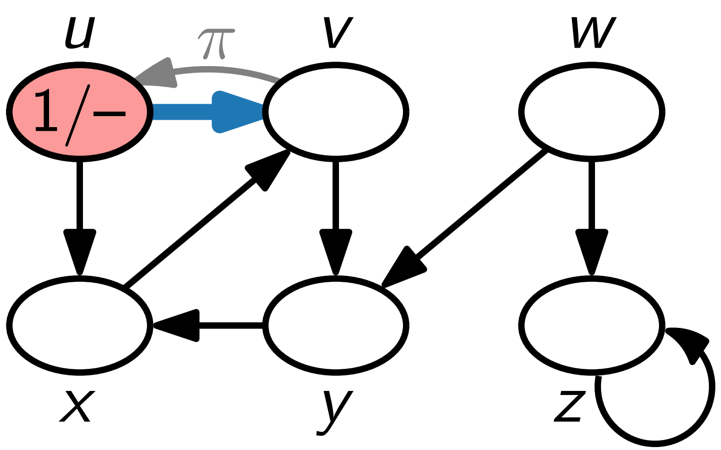


Tiefensuche

Eingabe: (un)gerichteter Graph G

Ausgabe:

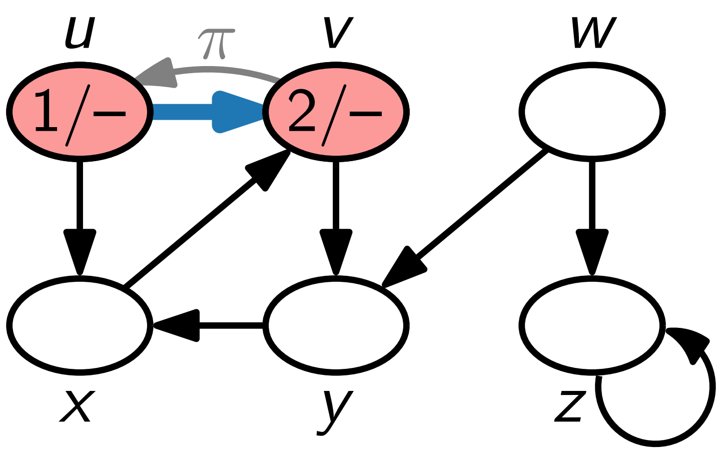
- Besuchsintervalle $(u.d/u.f)$ — discovery time, finish time
- DFS-Wald $\left(\overset{\pi}{\longleftarrow} \right)$



Tiefensuche

Eingabe: (un)gerichteter Graph G

Ausgabe:
- Besuchsintervalle $(u.d/u.f)$ — discovery time, finish time
- DFS-Wald $\left(\xleftarrow{\pi}\right)$



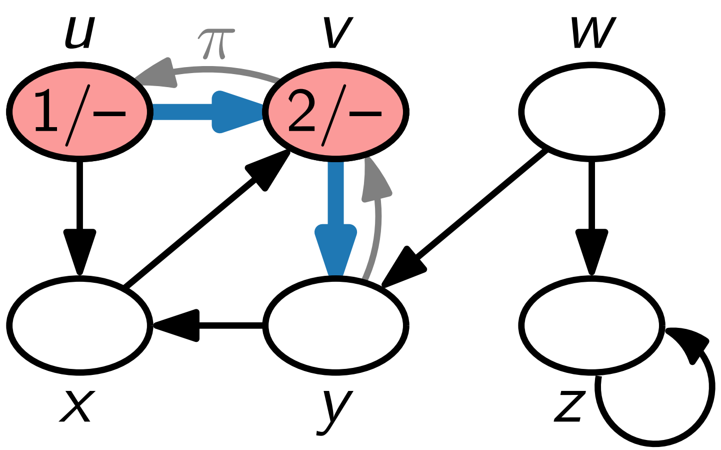
Tiefensuche

Eingabe: (un)gerichteter Graph G

discovery time

finish time

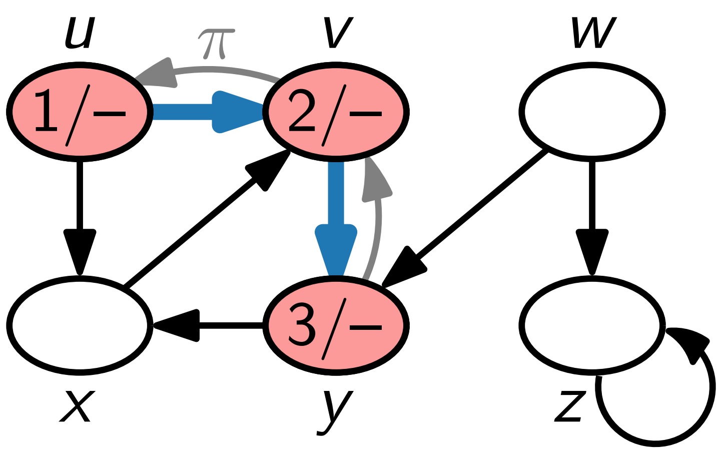
Ausgabe:
- Besuchsintervalle $(u.d/u.f)$
- DFS-Wald $\left(\xleftarrow{\pi}\right)$



Tiefensuche

Eingabe: (un)gerichteter Graph G

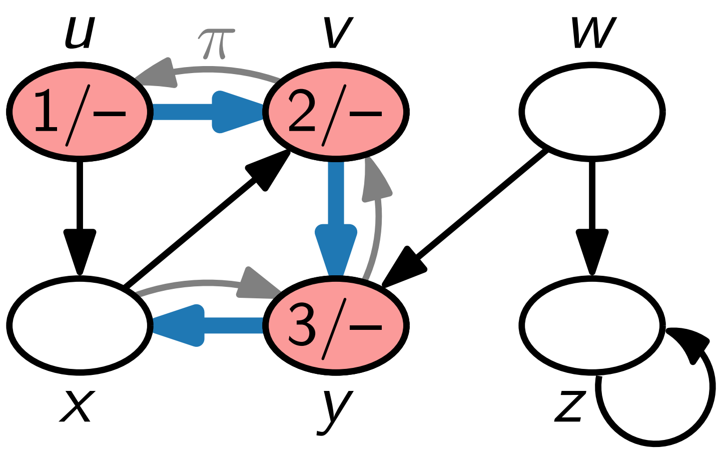
Ausgabe:
- Besuchsintervalle $(u.d/u.f)$ — discovery time / finish time
- DFS-Wald $\left(\xleftarrow{\pi}\right)$



Tiefensuche

Eingabe: (un)gerichteter Graph G

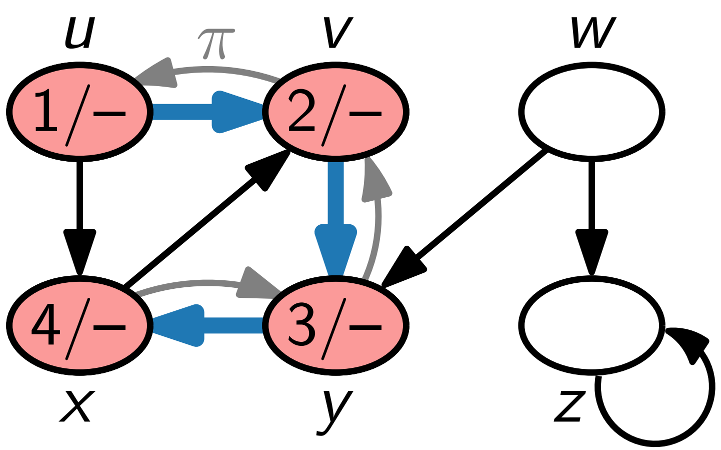
Ausgabe:
- Besuchsintervalle $(u.d/u.f)$ discovery time finish time
- DFS-Wald $\left(\xleftarrow{\pi}\right)$



Tiefensuche

Eingabe: (un)gerichteter Graph G

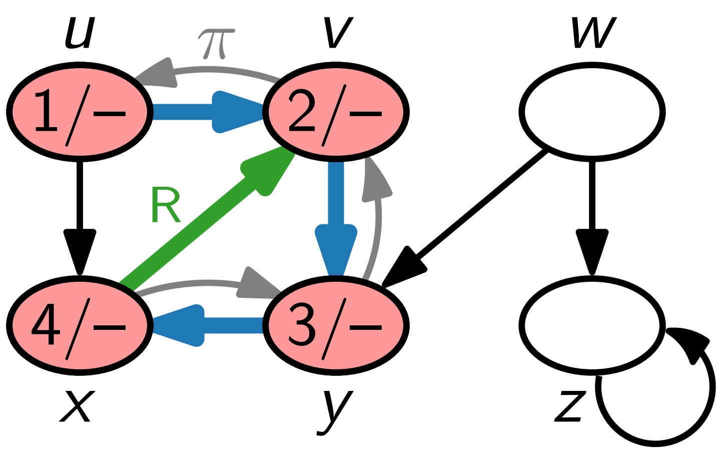
Ausgabe:
- Besuchsintervalle $(u.d/u.f)$ — discovery time: $u.d$, finish time: $u.f$
- DFS-Wald $\left(\xleftarrow{\pi}\right)$



Tiefensuche

Eingabe: (un)gerichteter Graph G

Ausgabe:
- Besuchsintervalle $(u.d/u.f)$ discovery time, finish time
- DFS-Wald (π)

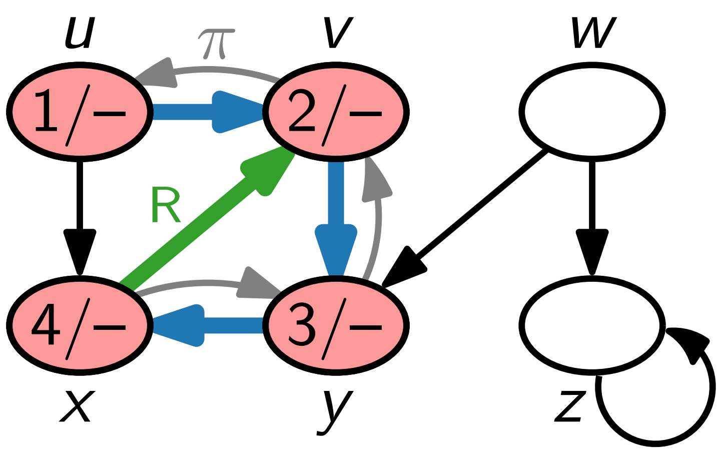


Tiefensuche

Eingabe: (un)gerichteter Graph G

Ausgabe:

- Besuchsintervalle $(u.d/u.f)$ — discovery time, finish time
- DFS-Wald $\left(\xleftarrow{\pi}\right)$
- Klassifizierung der Graphkanten:
  - Baumkanten (Kanten von G_π)
  - Rückwärtskanten (R)

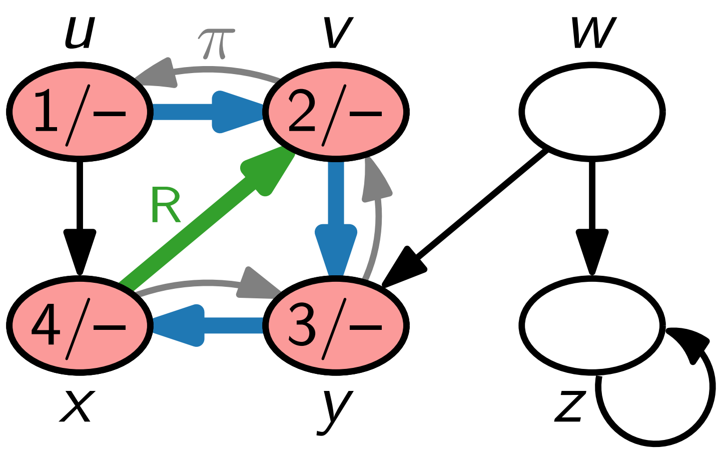


Farbe Zielknoten:

Tiefensuche

Eingabe: (un)gerichteter Graph G

Ausgabe:
- Besuchsintervalle $(u.d/u.f)$ discovery time finish time
- DFS-Wald $\left(\xleftarrow{\pi}\right)$
- Klassifizierung der Graphkanten:
  - Baumkanten (Kanten von G_π)
  - Rückwärtskanten (R)



Farbe Zielknoten:

weiß

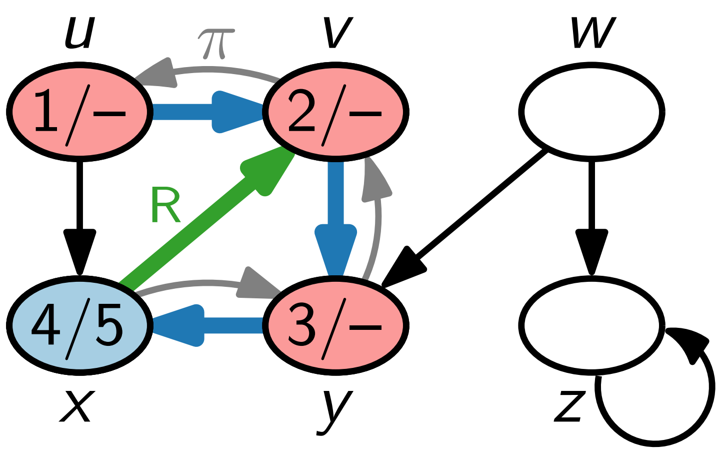
rot

Tiefensuche

Eingabe: (un)gerichteter Graph G

Ausgabe:

- Besuchsintervalle $(u.d/u.f)$ (discovery time, finish time)
- DFS-Wald $\left(\xleftarrow{\pi}\right)$
- Klassifizierung der Graphkanten:
  - Baumkanten (Kanten von G_π)
  - Rückwärtskanten (R)



Farbe Zielknoten:

weiß

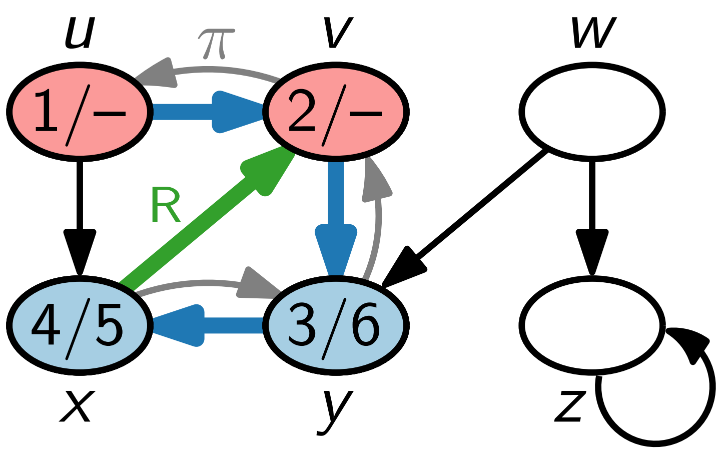
rot

Tiefensuche

Eingabe: (un)gerichteter Graph G

discovery time finish time

Ausgabe:
- Besuchsintervalle $(u.d/u.f)$
- DFS-Wald $\left(\xleftarrow{\pi}\right)$
- Klassifizierung der Graphkanten:
  - Baumkanten (Kanten von G_π)
  - Rückwärtskanten (R)



Farbe Zielknoten:

weiß

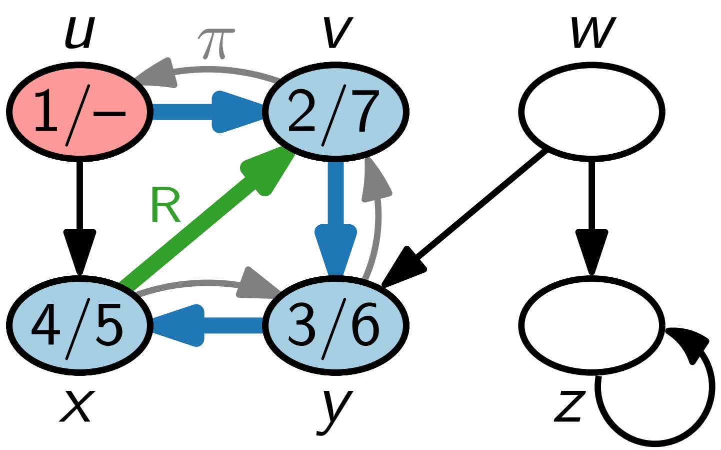
rot

Tiefensuche

Eingabe: (un)gerichteter Graph G

Ausgabe:

- Besuchsintervalle $(u.d/u.f)$ (discovery time / finish time)
- DFS-Wald $\left(\xleftarrow{\pi}\right)$
- Klassifizierung der Graphkanten:
  - Baumkanten (Kanten von G_π)
  - Rückwärtskanten (R)



Farbe Zielknoten:

weiß

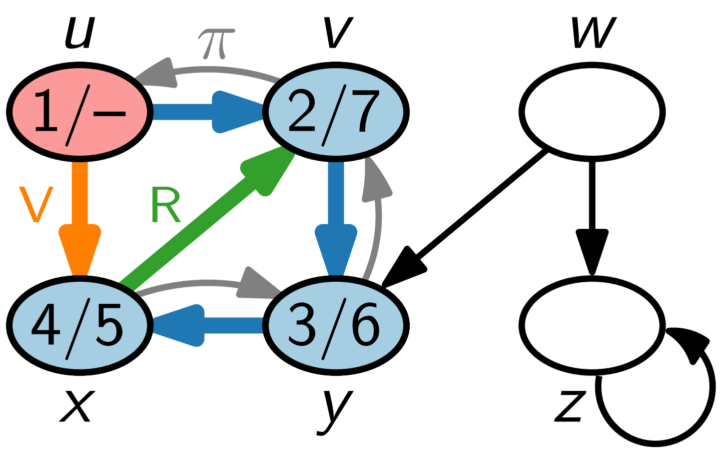
rot

Tiefensuche

Eingabe: (un)gerichteter Graph G

Ausgabe:

- Besuchsintervalle $(u.d/u.f)$ (discovery time / finish time)
- DFS-Wald (π)
- Klassifizierung der Graphkanten:
  - Baumkanten (Kanten von G_π)
  - Rückwärtskanten (R)



Farbe Zielknoten:

weiß

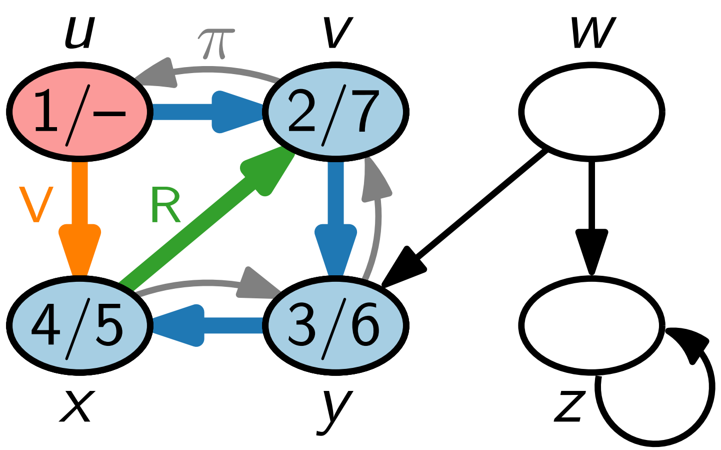
rot

Tiefensuche

Eingabe: (un)gerichteter Graph G

Ausgabe:

- Besuchsintervalle $(u.d/u.f)$ — discovery time, finish time
- DFS-Wald $(\xleftarrow{\pi})$
- Klassifizierung der Graphkanten:
  - Baumkanten (Kanten von G_π)
  - Rückwärtskanten (R)
  - Vorwärtskanten (V)



Farbe Zielknoten:

weiß

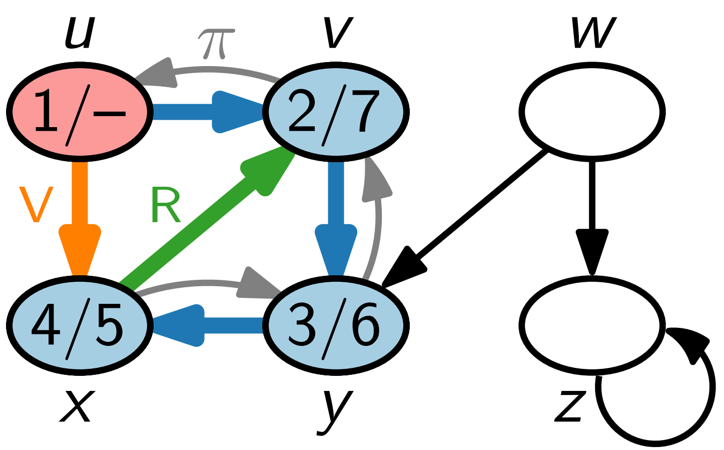
rot

Tiefensuche

Eingabe: (un)gerichteter Graph G

Ausgabe:

- Besuchsintervalle $(u.d/u.f)$ — discovery time, finish time
- DFS-Wald $(\xleftarrow{\pi})$
- Klassifizierung der Graphkanten:
  - Baumkanten (Kanten von G_π)
  - Rückwärtskanten (R)
  - Vorwärtskanten (V)



Farbe Zielknoten:

weiß

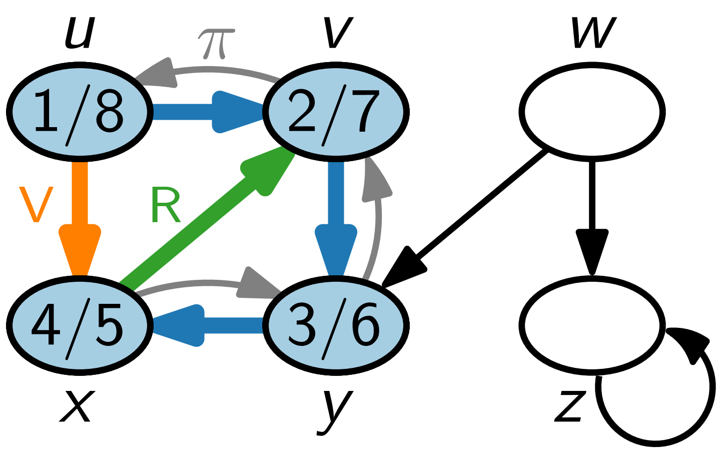
rot

blau

Tiefensuche

Eingabe: (un)gerichteter Graph G

Ausgabe:
- Besuchsintervalle $(u.d / u.f)$ (discovery time, finish time)
- DFS-Wald $\left(\overset{\pi}{\longleftarrow} \right)$
- Klassifizierung der Graphkanten:
  - Baumkanten (Kanten von G_π)
  - Rückwärtskanten (R)
  - Vorwärtskanten (V)



Farbe Zielknoten:

weiß

rot

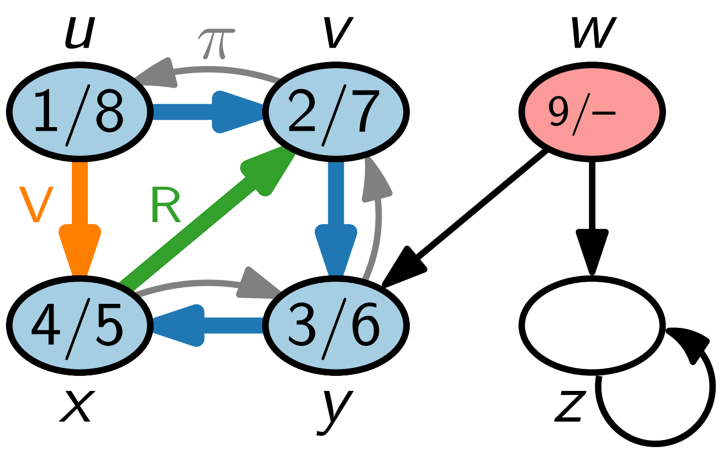
blau

Tiefensuche

Eingabe: (un)gerichteter Graph G

Ausgabe:

- Besuchsintervalle $(u.d/u.f)$ discovery time finish time
- DFS-Wald $\left(\xleftarrow{\pi}\right)$
- Klassifizierung der Graphkanten:
  - Baumkanten (Kanten von G_π)
  - Rückwärtskanten (R)
  - Vorwärtskanten (V)



Farbe Zielknoten:

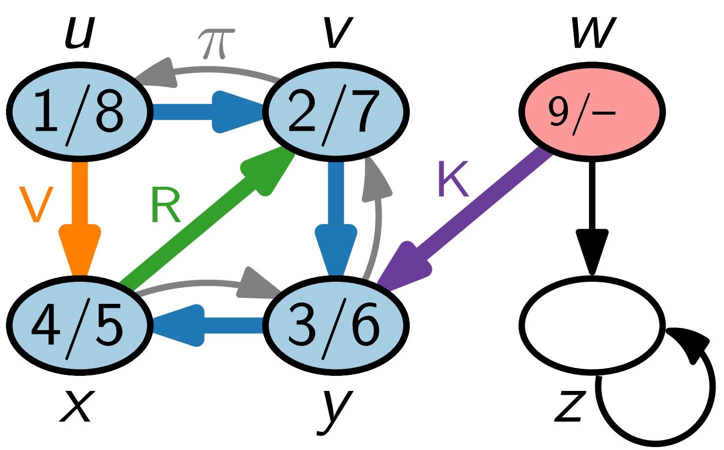
weiß

rot

blau

Tiefensuche

Eingabe: (un)gerichteter Graph G



Ausgabe:

- Besuchsintervalle $(u.d/u.f)$ discovery time finish time
- DFS-Wald $(\overset{\pi}{\leftarrow})$
- Klassifizierung der Graphkanten:
  - Baumkanten (Kanten von G_π)
  - Rückwärtskanten (R)
  - Vorwärtskanten (V)

Farbe Zielknoten:

weiß

rot

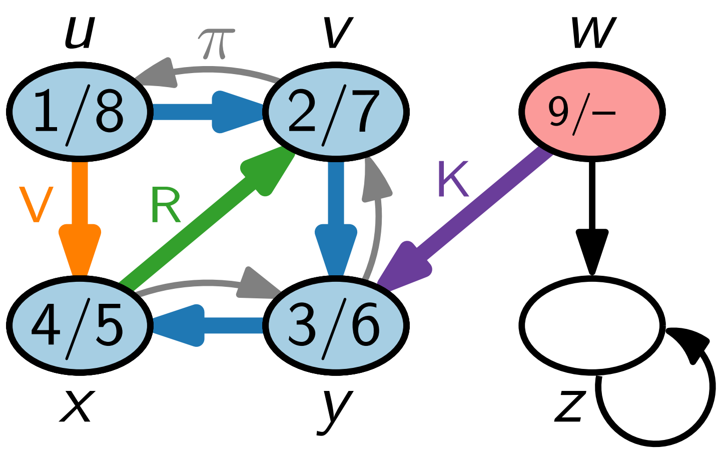
blau

Tiefensuche

Eingabe: (un)gerichteter Graph G

Ausgabe:

- Besuchsintervalle $(u.d/u.f)$ (discovery time / finish time)
- DFS-Wald $\left(\xleftarrow{\pi}\right)$
- Klassifizierung der Graphkanten:
  - Baumkanten (Kanten von G_π)
  - Rückwärtskanten (R)
  - Vorwärtskanten (V)
  - Kreuzkanten (K)



Farbe Zielknoten:

weiß

rot

blau

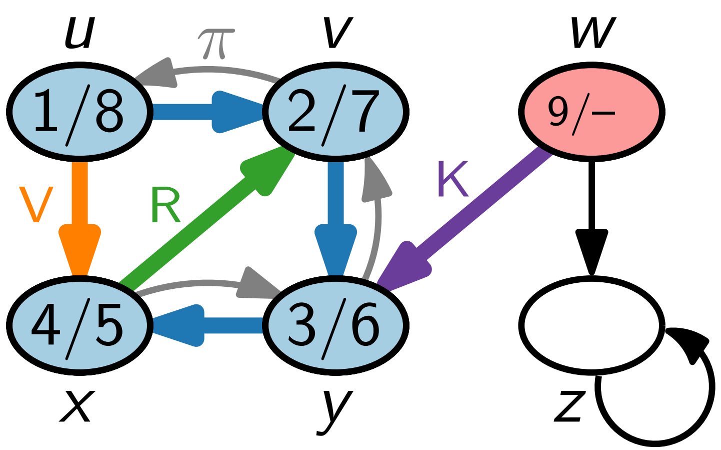
Tiefensuche

Eingabe: (un)gerichteter Graph G

Ausgabe:

discovery time · finish time

- Besuchsintervalle $(u.d/u.f)$
- DFS-Wald $\left(\xleftarrow{\pi}\right)$
- Klassifizierung der Graphkanten:
  - Baumkanten (Kanten von G_π)
  - Rückwärtskanten (R)
  - Vorwärtskanten (V)
  - Kreuzkanten (K)



Farbe Zielknoten:

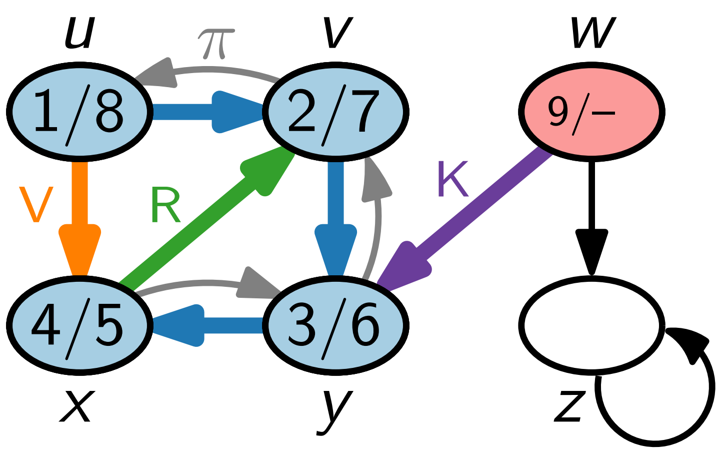
weiß

rot

blau

blau

Tiefensuche



Eingabe: (un)gerichteter Graph G

Ausgabe:

- Besuchsintervalle $(u.d/u.f)$ (discovery time / finish time)
- DFS-Wald (π)
- Klassifizierung der Graphkanten:
  - Baumkanten (Kanten von G_π)
  - Rückwärtskanten (R)
  - Vorwärtskanten (V)
  - Kreuzkanten (K)

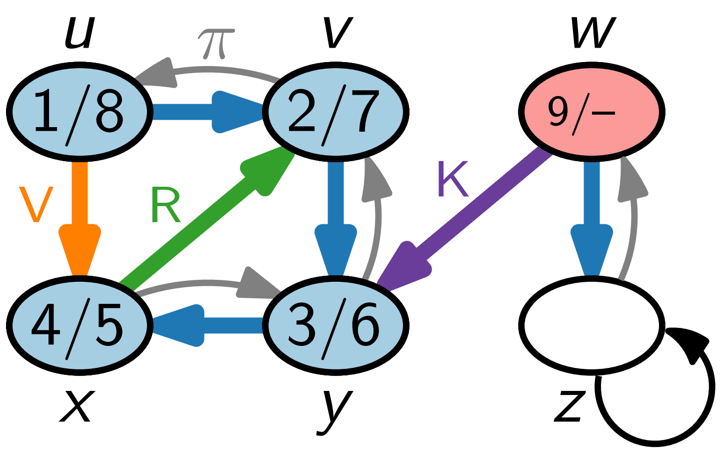
	Farbe Zielknoten:
Baumkanten (Kanten von G_π)	weiß
Rückwärtskanten (R)	rot
Vorwärtskanten (V)	blau und start.d $<$ ziel.d
Kreuzkanten (K)	blau und start.d $>$ ziel.d

Tiefensuche

Eingabe: (un)gerichteter Graph G

Ausgabe:

- Besuchsintervalle $(u.d/u.f)$ (discovery time / finish time)
- DFS-Wald $\left(\xleftarrow{\pi}\right)$
- Klassifizierung der Graphkanten:
  - Baumkanten (Kanten von G_π)
  - Rückwärtskanten (R)
  - Vorwärtskanten (V)
  - Kreuzkanten (K)



Farbe Zielknoten:

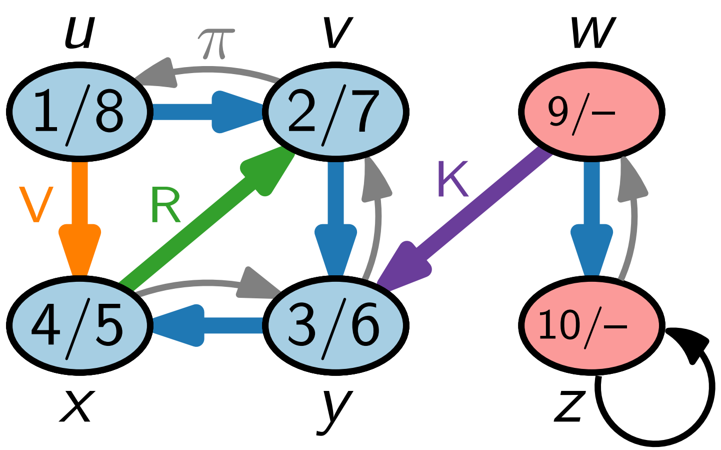
	Farbe Zielknoten
Baumkanten	weiß
Rückwärtskanten (R)	rot
Vorwärtskanten (V)	blau und start.$d <$ ziel.d
Kreuzkanten (K)	blau und start.$d >$ ziel.d

Tiefensuche

Eingabe: (un)gerichteter Graph G

Ausgabe:

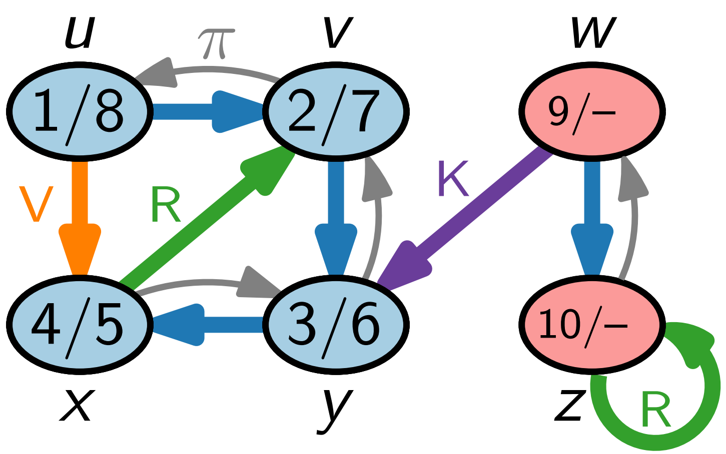
- Besuchsintervalle $(u.d/u.f)$ — discovery time / finish time
- DFS-Wald $(\overset{\pi}{\leftarrow})$
- Klassifizierung der Graphkanten:
  - Baumkanten (Kanten von G_π)
  - Rückwärtskanten (R)
  - Vorwärtskanten (V)
  - Kreuzkanten (K)



Farbe Zielknoten:

	Farbe Zielknoten
Baumkanten	weiß
Rückwärtskanten	rot
Vorwärtskanten	blau und start.$d <$ ziel.d
Kreuzkanten	blau und start.$d >$ ziel.d

Tiefensuche



Eingabe: (un)gerichteter Graph G

Ausgabe:

- Besuchsintervalle $(u.d / u.f)$ — discovery time, finish time
- DFS-Wald $\left(\xleftarrow{\pi}\right)$
- Klassifizierung der Graphkanten:
  - Baumkanten (Kanten von G_π)
  - Rückwärtskanten (R)
  - Vorwärtskanten (V)
  - Kreuzkanten (K)

Farbe Zielknoten:

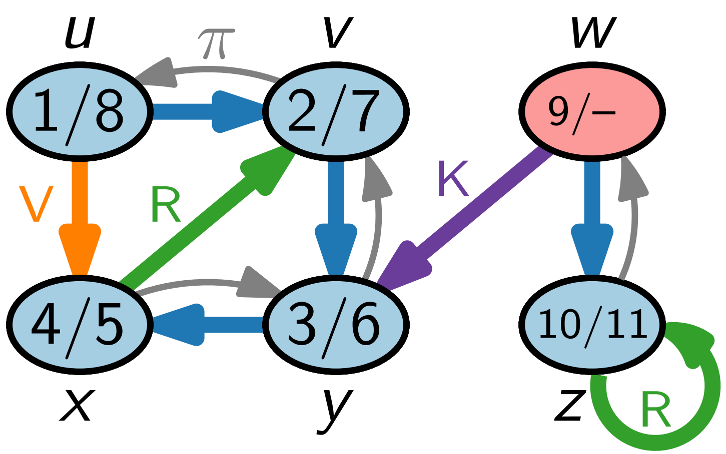
weiß

rot

blau und start.d < ziel.d

blau und start.d > ziel.d

Tiefensuche



Eingabe: (un)gerichteter Graph G

Ausgabe:
- Besuchsintervalle $(u.d/u.f)$ — discovery time, finish time
- DFS-Wald $\left(\xleftarrow{\pi}\right)$
- Klassifizierung der Graphkanten:
  - Baumkanten (Kanten von G_π)
  - Rückwärtskanten (R)
  - Vorwärtskanten (V)
  - Kreuzkanten (K)

Farbe Zielknoten:

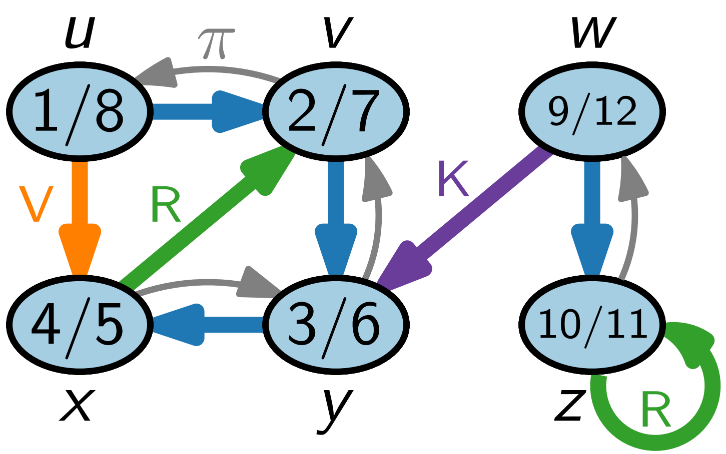
weiß

rot

blau und start.d < ziel.d

blau und start.d > ziel.d

Tiefensuche



Eingabe: (un)gerichteter Graph G

Ausgabe:
- Besuchsintervalle $(u.d/u.f)$ discovery time, finish time
- DFS-Wald $(\xleftarrow{\pi})$
- Klassifizierung der Graphkanten:
  - Baumkanten (Kanten von G_π)
  - Rückwärtskanten (R)
  - Vorwärtskanten (V)
  - Kreuzkanten (K)

Farbe Zielknoten:

weiß

rot

blau und start.d $<$ ziel.d

blau und start.d $>$ ziel.d

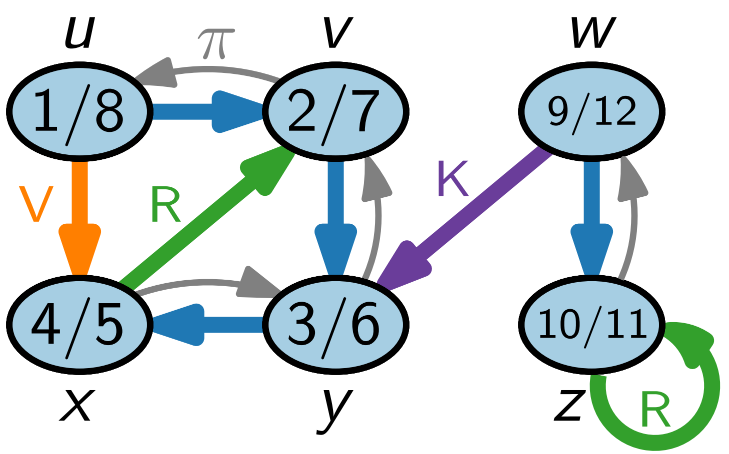
Tiefensuche

Eingabe: (un)gerichteter Graph G

Ausgabe:

- Besuchsintervalle $(u.d/u.f)$ — discovery time / finish time
- DFS-Wald $\left(\xleftarrow{\pi}\right)$
- Klassifizierung der Graphkanten:
  - Baumkanten (Kanten von G_π)

    Kanten des DFS-Waldes (entgegen π gerichtet)
  - Rückwärtskanten (R)
  - Vorwärtskanten (V)
  - Kreuzkanten (K)



Farbe Zielknoten:

weiß

rot

blau und start.$d <$ ziel.d

blau und start.$d >$ ziel.d

Tiefensuche

Eingabe: (un)gerichteter Graph G

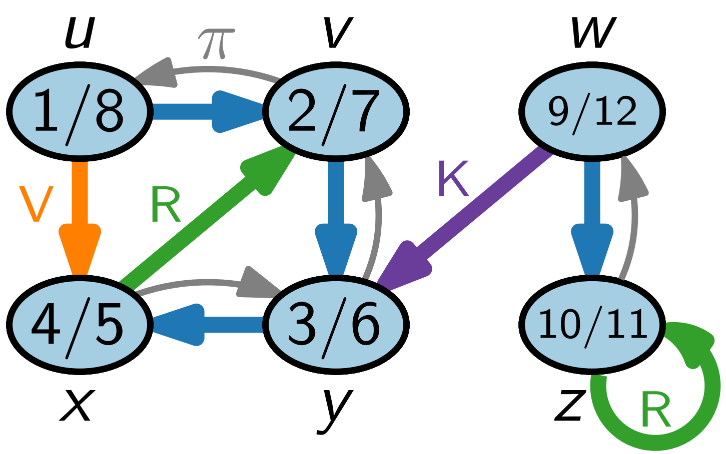
Ausgabe:

discovery time · finish time

- Besuchsintervalle $(u.d/u.f)$
- DFS-Wald $\left(\overset{\pi}{\longleftarrow}\right)$
- Klassifizierung der Graphkanten:
  - Baumkanten (Kanten von G_π)

    Kanten des DFS-Waldes (entgegen π gerichtet)
  - Rückwärtskanten (R)

    Nicht-Baumkanten zu einem Vorgängerknoten
  - Vorwärtskanten (V)
  - Kreuzkanten (K)



Farbe Zielknoten:

weiß

rot

blau und start.d < ziel.d

blau und start.d > ziel.d

Tiefensuche

Eingabe: (un)gerichteter Graph G

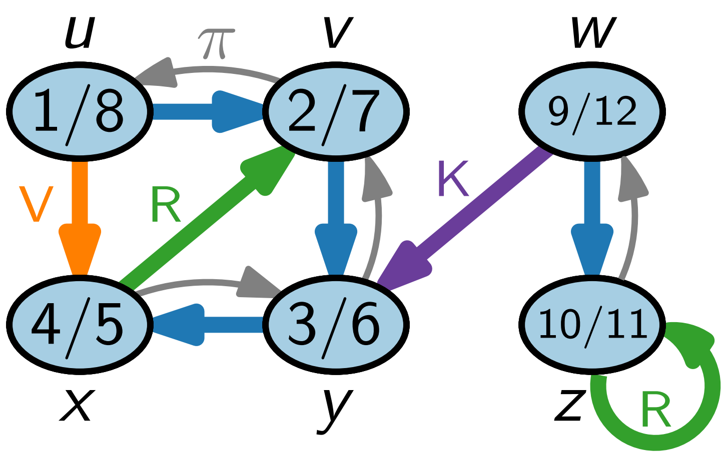
Ausgabe:

- Besuchsintervalle $(u.d/u.f)$ (discovery time, finish time)
- DFS-Wald $\left(\xleftarrow{\pi}\right)$
- Klassifizierung der Graphkanten:
  - Baumkanten (Kanten von G_π)

    Kanten des DFS-Waldes (entgegen π gerichtet)
  - Rückwärtskanten (R)

    Nicht-Baumkanten zu einem Vorgängerknoten
  - Vorwärtskanten (V)

    Nicht-Baumkanten zu einem Nachfolgerknoten
  - Kreuzkanten (K)



Farbe Zielknoten:

weiß

rot

blau und start.$d <$ ziel.d

blau und start.$d >$ ziel.d

Tiefensuche

Eingabe: (un)gerichteter Graph G

Ausgabe:

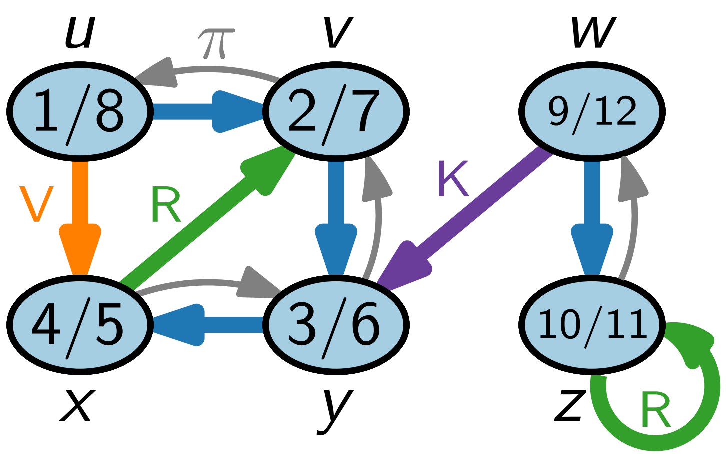
- Besuchsintervalle $(u.d/u.f)$ (discovery time / finish time)
- DFS-Wald $\left(\xleftarrow{\pi}\right)$
- Klassifizierung der Graphkanten:
  - Baumkanten (Kanten von G_π)

    Kanten des DFS-Waldes (entgegen π 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:

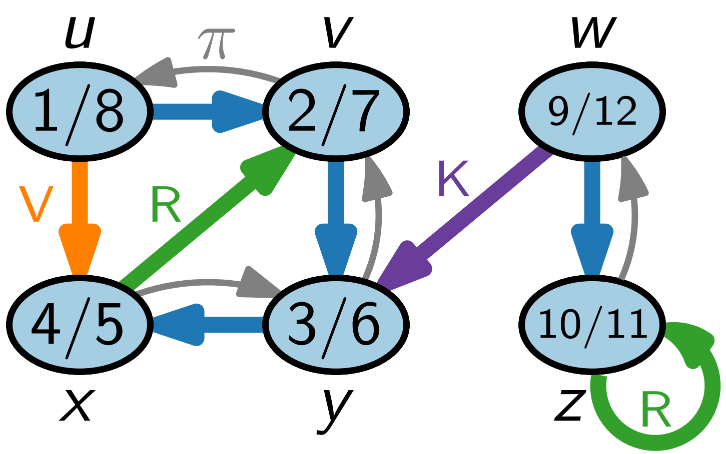
weiß

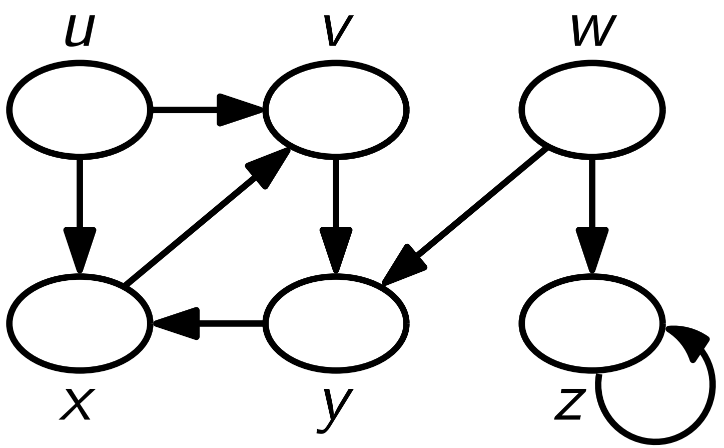
rot

blau und start.$d <$ ziel.d

blau und start.$d >$ ziel.d

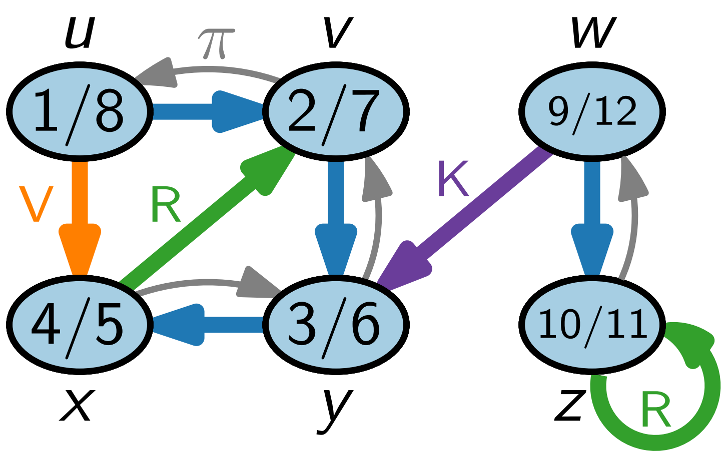
Tiefensuche – Pseudocode

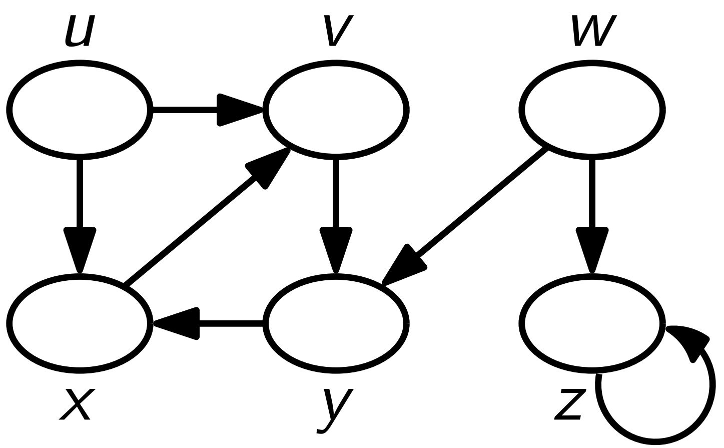




Tiefensuche – Pseudocode

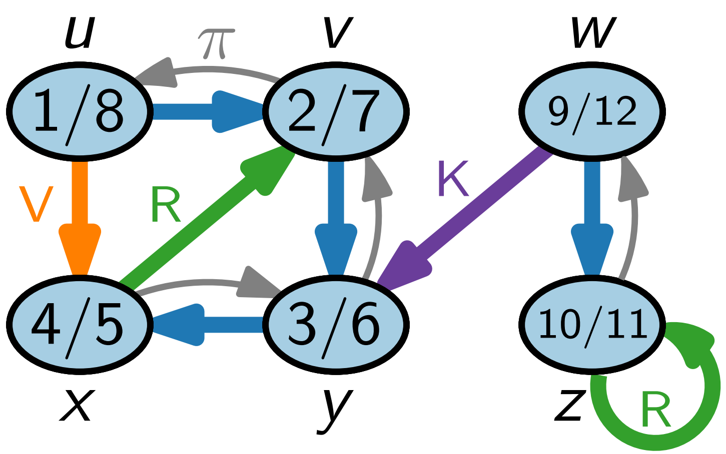
DFS(Graph G)

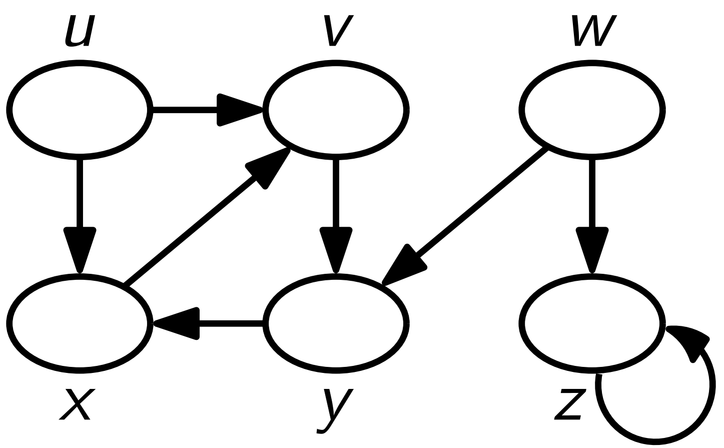




Tiefensuche – Pseudocode

DFS(Graph G)
foreach $u \in V(G)$ **do**



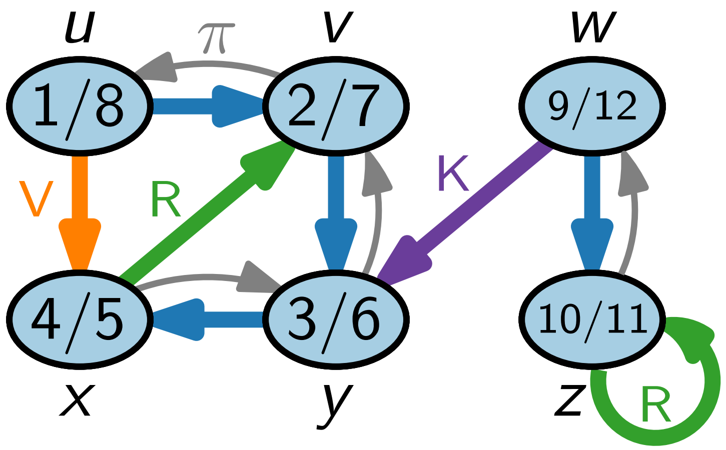


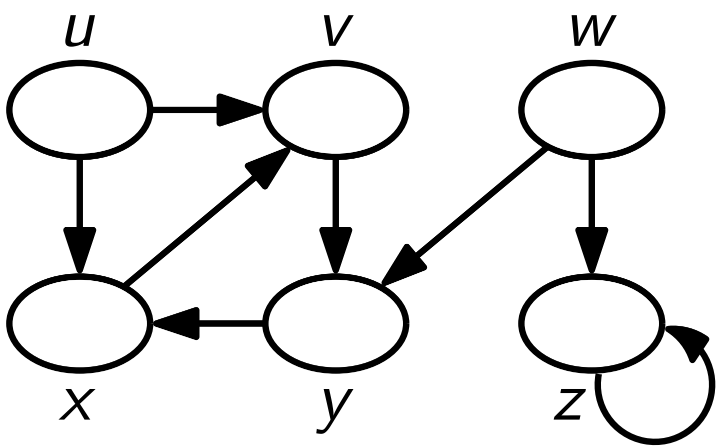
Tiefensuche – Pseudocode

DFS(Graph G)

foreach $u \in V(G)$ **do**

  $u.color = white$





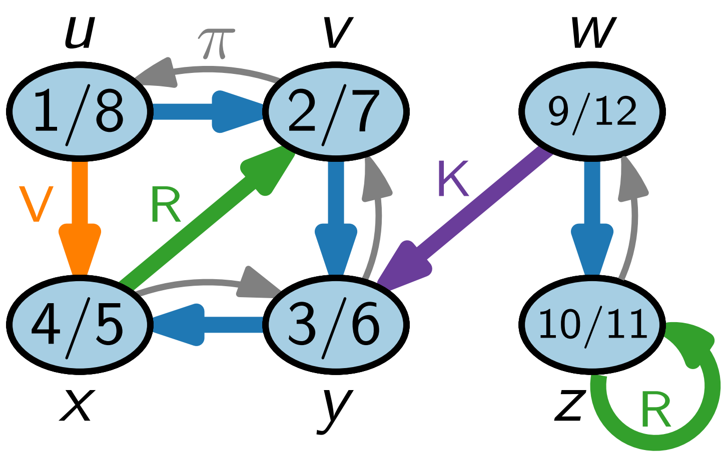
Tiefensuche – Pseudocode

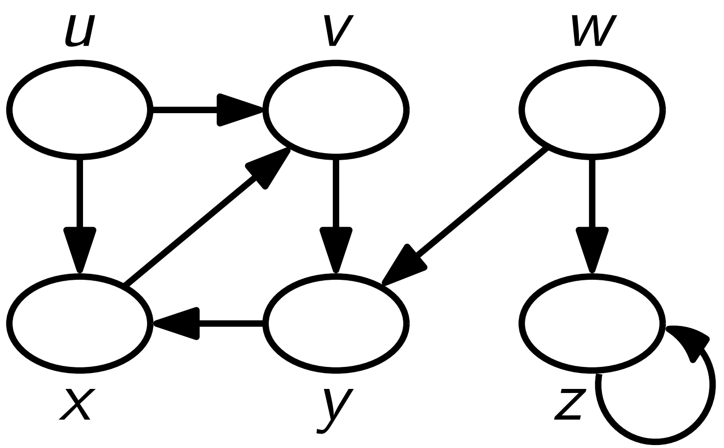
DFS(Graph G)

foreach $u \in V(G)$ **do**

$\quad u.color = white$

$\quad u.\pi = nil$





Tiefensuche – Pseudocode

DFS(Graph G)

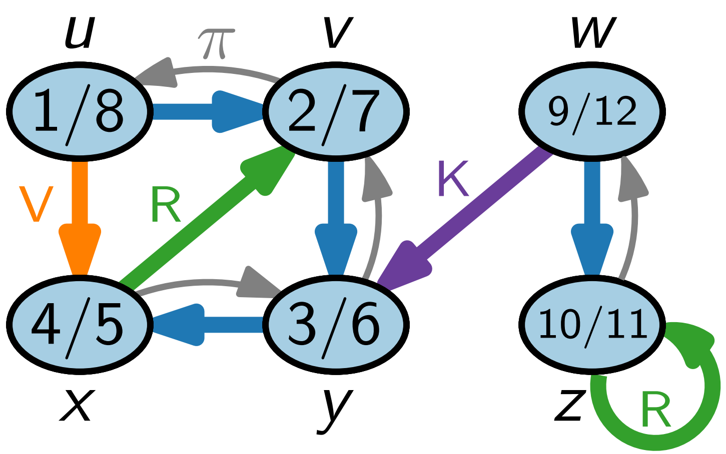
foreach $u \in V(G)$ **do**

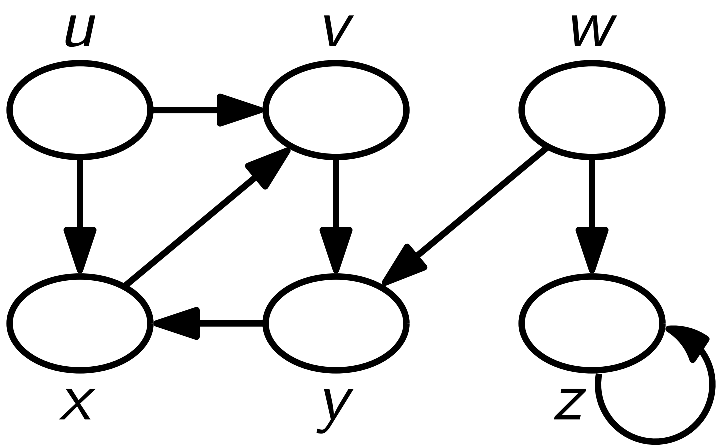
  $u.color = white$

  $u.\pi = nil$

$time = 0$ globale Variable

$time = 0$

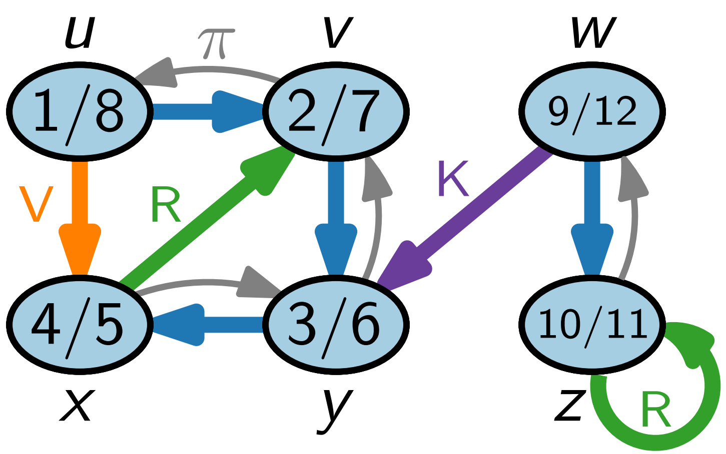


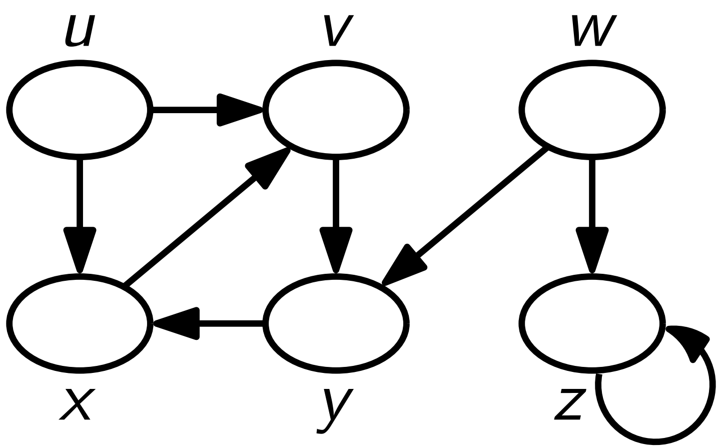


Tiefensuche – Pseudocode

DFS(Graph G)
  foreach $u \in V(G)$ **do**
    $u.color = white$
    $u.\pi = nil$
  $time = 0$ globale Variable
  foreach $u \in V(G)$ **do**

$\boxed{time = 0}$

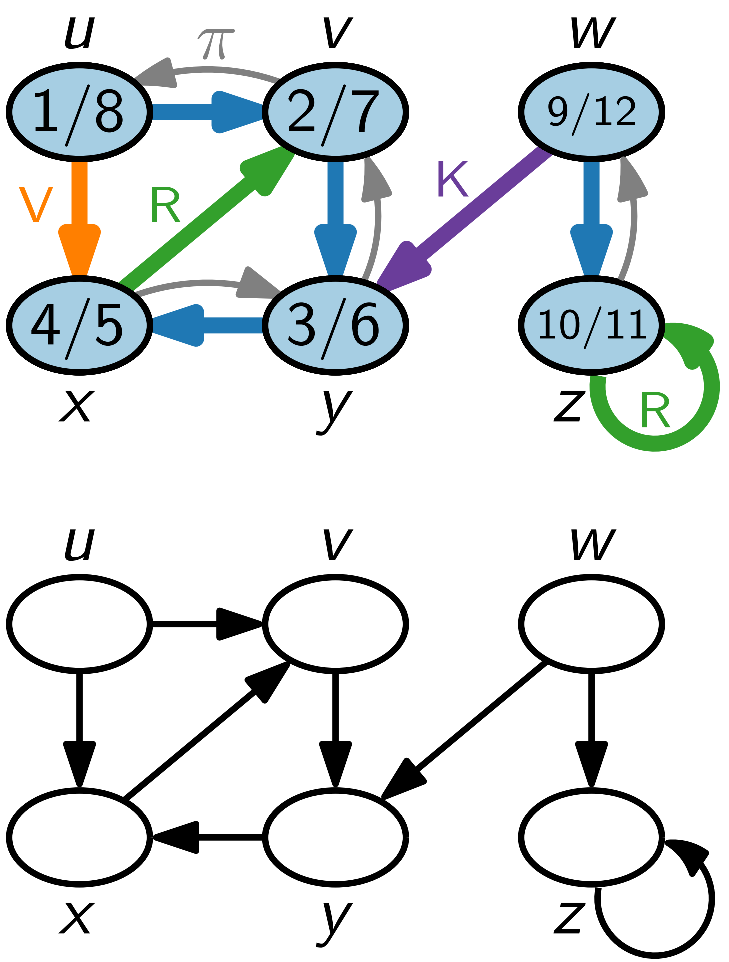




Tiefensuche – Pseudocode

DFS(Graph G)
foreach $u \in V(G)$ **do**
  $u.color = white$
  $u.\pi = nil$
$time = 0$ globale Variable
foreach $u \in V(G)$ **do**
  if $u.color == white$ **then**

$\boxed{time = 0}$



Tiefensuche – Pseudocode

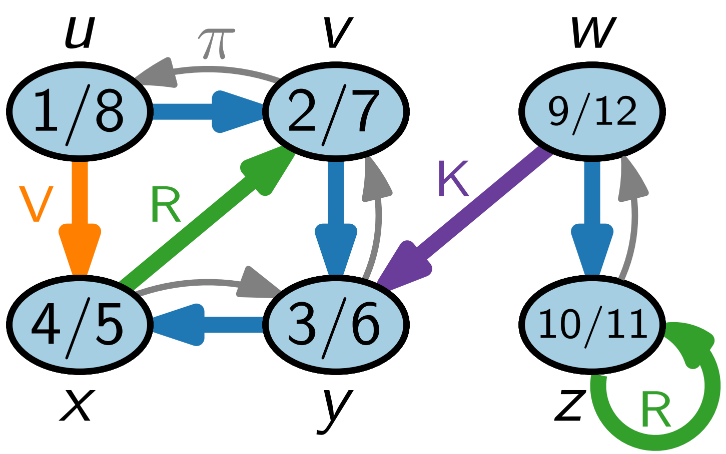
DFS(Graph G)

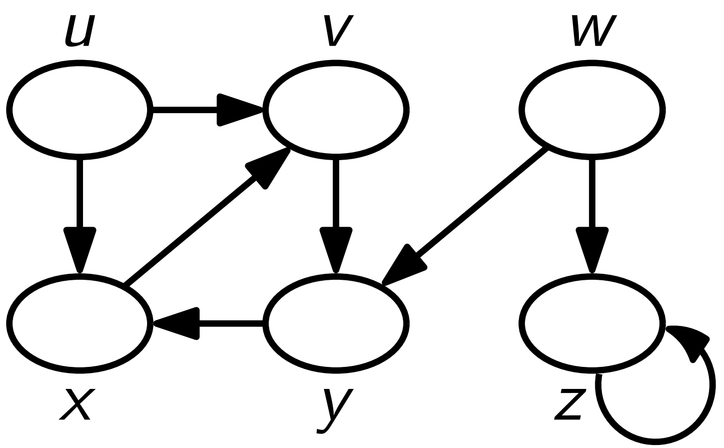
foreach $u \in V(G)$ **do**
- $u.color = white$
- $u.\pi = nil$

$time = 0$ globale Variable

foreach $u \in V(G)$ **do**
- **if** $u.color == white$ **then** $\mathrm{DFSVisit}(G, u)$

$\boxed{time = 0}$



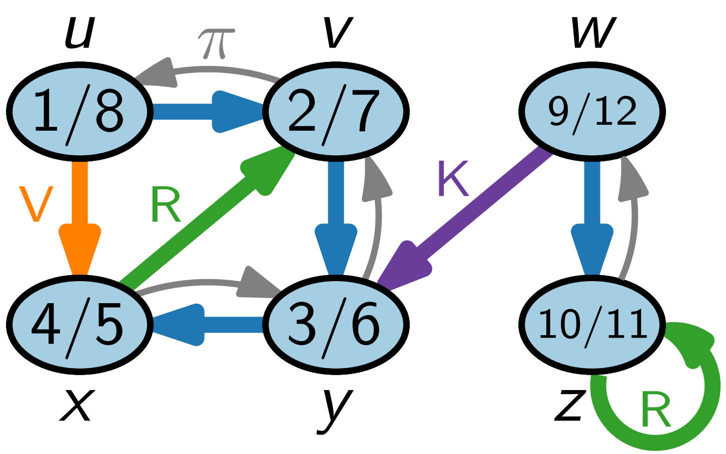


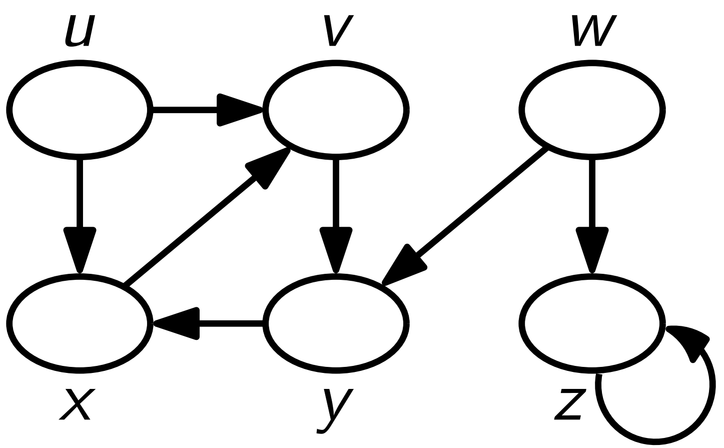
Tiefensuche – Pseudocode

DFS(Graph G)
foreach $u \in V(G)$ **do**
  $u.color = white$
  $u.\pi = nil$
$time = 0$ globale Variable
foreach $u \in V(G)$ **do**
  if $u.color == white$ **then** DFSVISIT(G, u)

DFSVISIT(Graph G, Vertex u)

$time = 0$



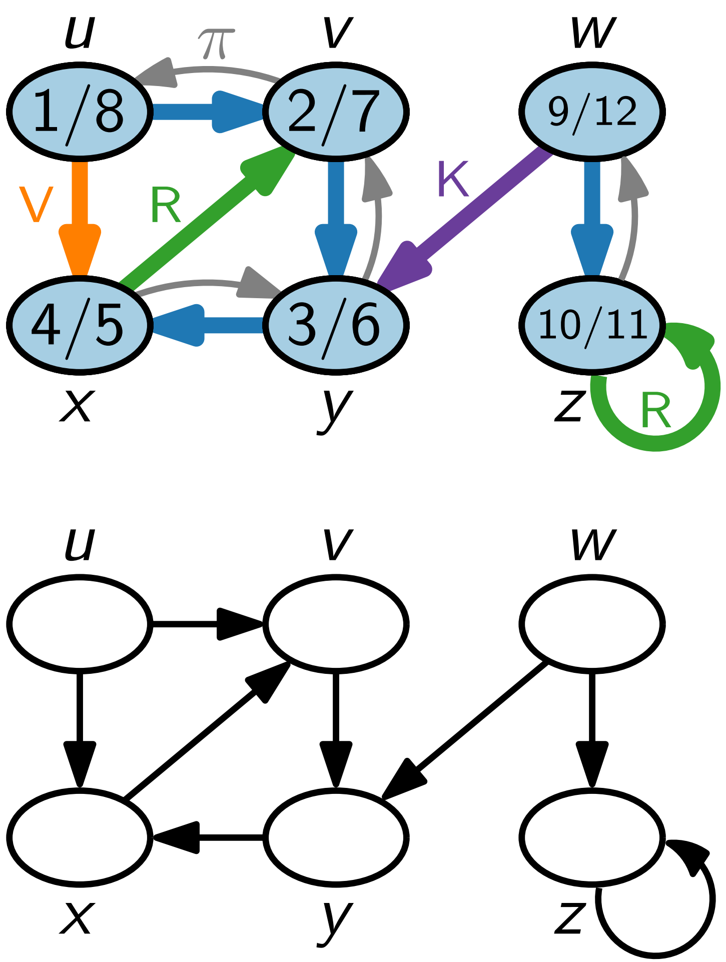


Tiefensuche – Pseudocode

DFS(Graph G)
  foreach $u \in V(G)$ **do**
    $u.color = white$
    $u.\pi = nil$
  $time = 0$ globale Variable
  foreach $u \in V(G)$ **do**
    if $u.color == white$ **then** DFSVISIT(G, u)

DFSVISIT(Graph G, Vertex u)
  $time = time + 1$

$time = 1$

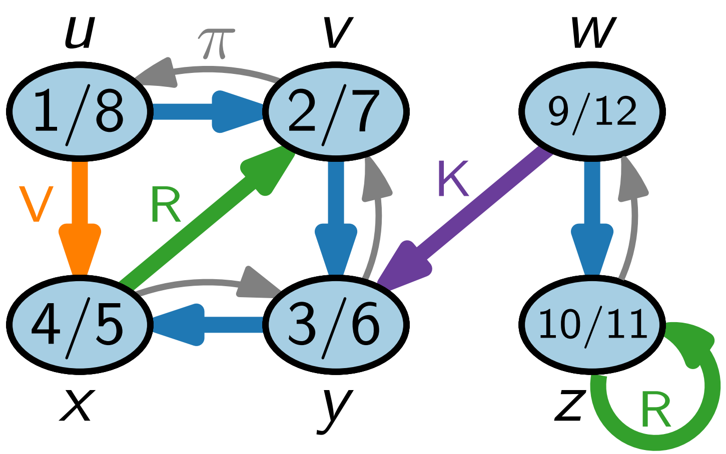


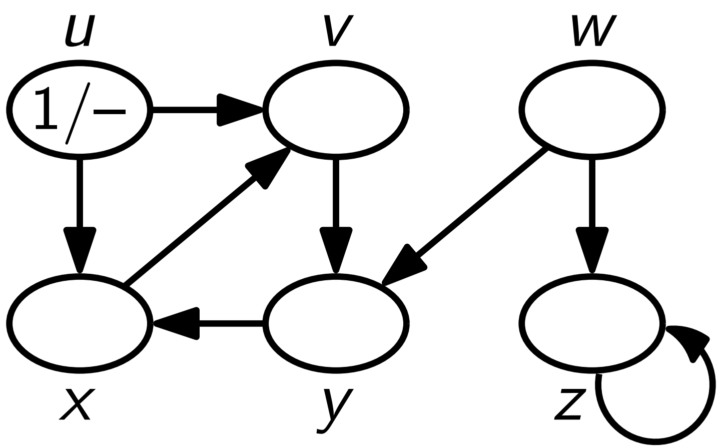
Tiefensuche – Pseudocode

DFS(Graph G)
  foreach $u \in V(G)$ **do**
    $u.color = white$
    $u.\pi = nil$
  $time = 0$ globale Variable
  foreach $u \in V(G)$ **do**
    if $u.color == white$ **then** DFSVisit(G, u)

DFSVisit(Graph G, Vertex u)
  $time = time + 1$
  $u.d = time;$

$time = 1$



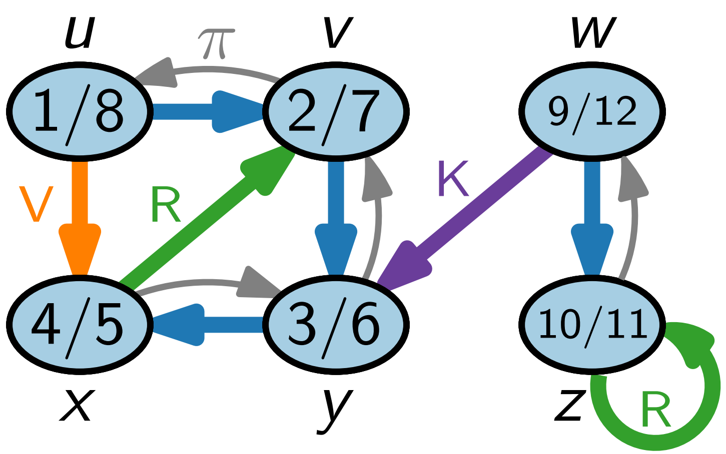


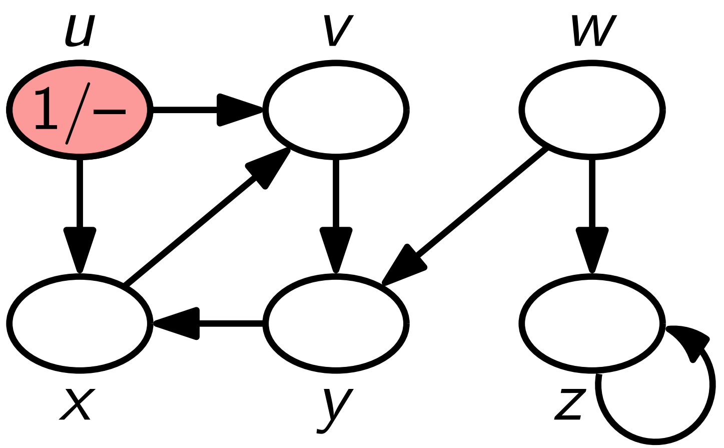
Tiefensuche – Pseudocode

DFS(Graph G)
  foreach $u \in V(G)$ **do**
    $u.color = white$
    $u.\pi = nil$
  $time = 0$ globale Variable
  foreach $u \in V(G)$ **do**
    if $u.color == white$ **then** DFSVisit(G, u)

DFSVisit(Graph G, Vertex u)
  $time = time + 1$
  $u.d = time$; $u.color = red$

$time = 1$



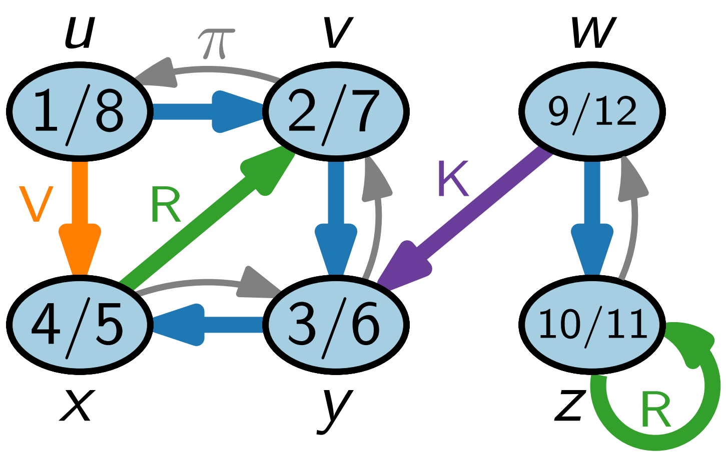


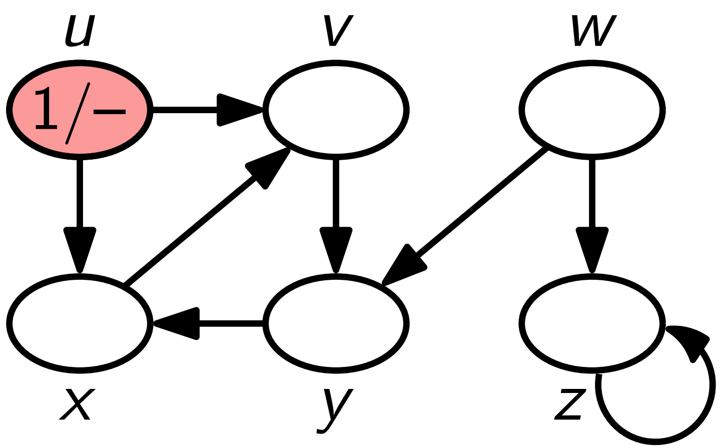
Tiefensuche – Pseudocode

DFS(Graph G)
  foreach $u \in V(G)$ **do**
    $u.color = white$
    $u.\pi = nil$
  $time = 0$ globale Variable
  foreach $u \in V(G)$ **do**
    if $u.color == white$ **then** DFSVISIT(G, u)

DFSVISIT(Graph G, Vertex u)
  $time = time + 1$
  $u.d = time$; $u.color = red$

$\boxed{time = 1}$





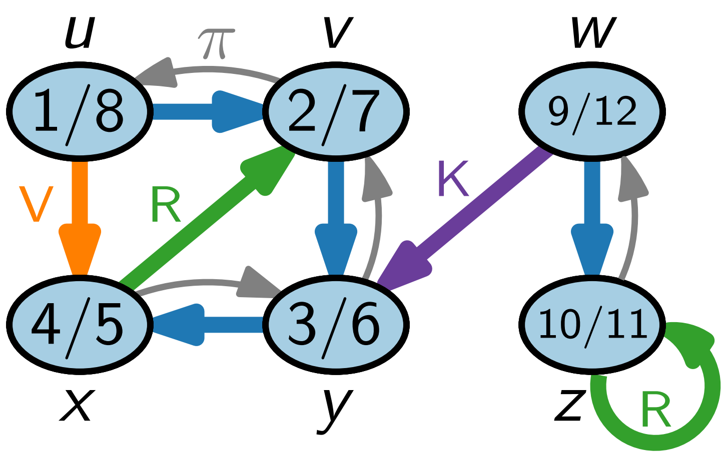
Für jeden Knoten u von G ist

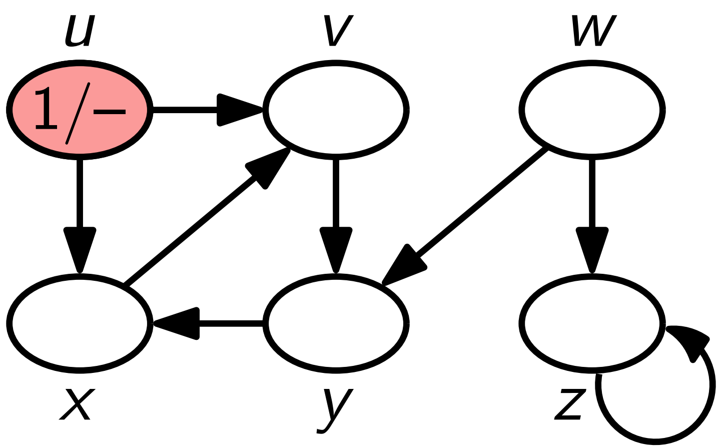
Tiefensuche – Pseudocode

DFS(Graph G)
  foreach $u \in V(G)$ **do**
    $u.color = white$
    $u.\pi = nil$
  $time = 0$ globale Variable
  foreach $u \in V(G)$ **do**
    if $u.color == white$ **then** DFSVisit(G, u)

DFSVisit(Graph G, Vertex u)
  $time = time + 1$
  $u.d = time$; $u.color = red$

$time = 1$





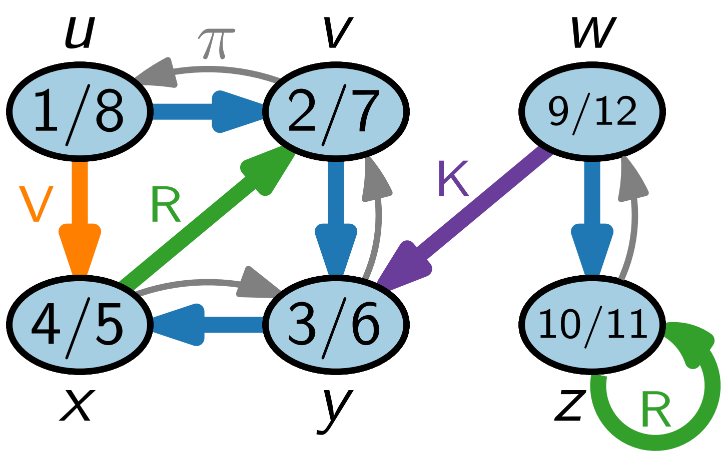
Für jeden Knoten u von G ist
- $u.d$ der Zeitpunkt der Entdeckung,

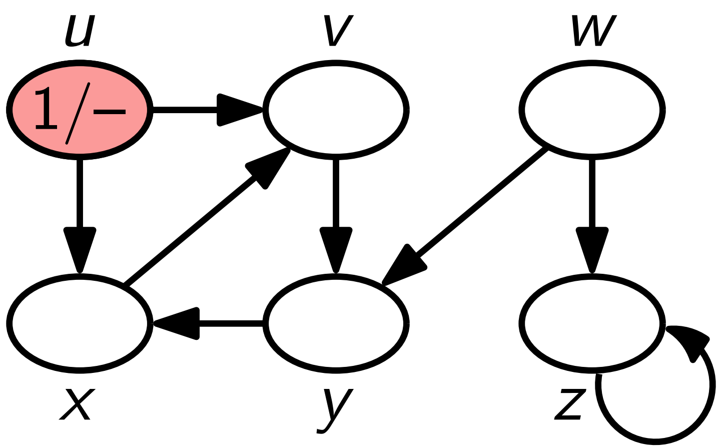
Tiefensuche – Pseudocode

DFS(Graph G)
foreach $u \in V(G)$ **do**
  $u.color = white$
  $u.\pi = nil$
$time = 0$ globale Variable
foreach $u \in V(G)$ **do**
  if $u.color == white$ **then** DFSVISIT(G, u)

DFSVISIT(Graph G, Vertex u)
$time = time + 1$
$u.d = time$; $u.color = red$

$time = 1$





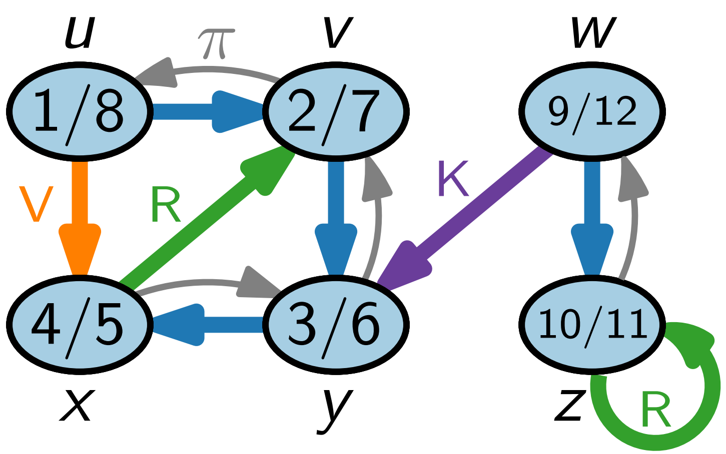
Für jeden Knoten u von G ist
- $u.d$ der Zeitpunkt der Entdeckung,
- $u.f$ der Abschluss-Zeitpunkt;

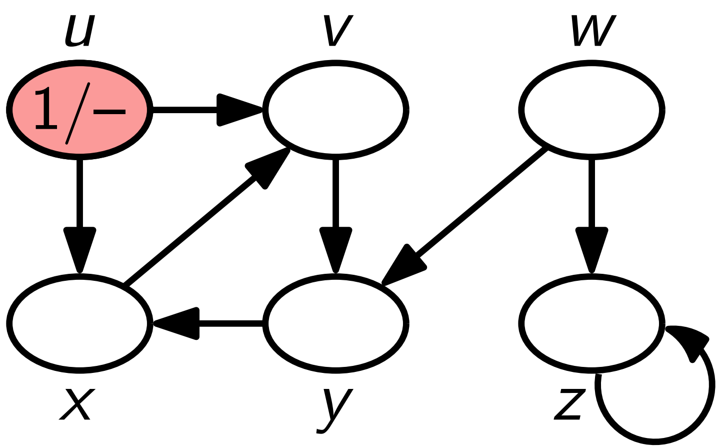
Tiefensuche – Pseudocode

DFS(Graph G)
foreach $u \in V(G)$ **do**
  $u.color = white$
  $u.\pi = nil$
$time = 0$ globale Variable
foreach $u \in V(G)$ **do**
  if $u.color == white$ **then** DFSVISIT(G, u)

DFSVISIT(Graph G, Vertex u)
$time = time + 1$
$u.d = time$; $u.color = red$

$time = 1$





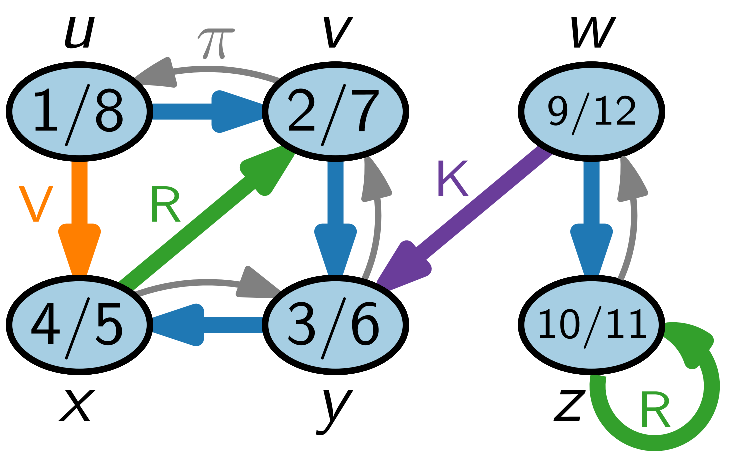
Für jeden Knoten u von G ist
- $u.d$ der Zeitpunkt der Entdeckung,
- $u.f$ der Abschluss-Zeitpunkt;

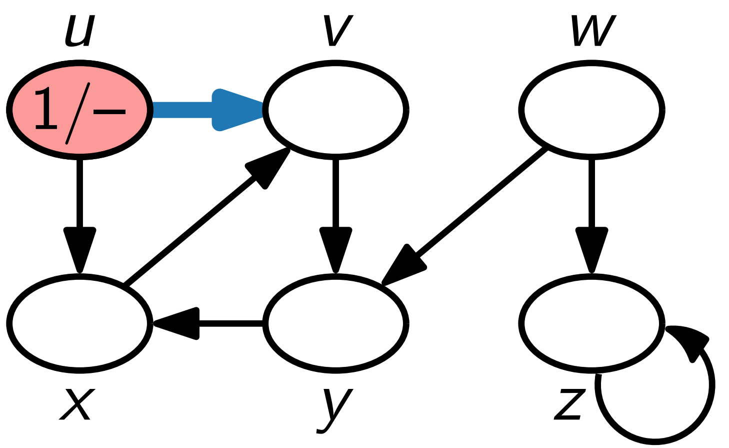
Besuchsintervall von u ist $[u.d, u.f]$.

Tiefensuche – Pseudocode

DFS(Graph G)
  foreach $u \in V(G)$ **do**
    $u.color = white$
    $u.\pi = nil$
  $time = 0$ globale Variable
  foreach $u \in V(G)$ **do**
    if $u.color == white$ **then** DFSVisit(G, u)

DFSVisit(Graph G, Vertex u) $time = 1$
  $time = time + 1$
  $u.d = time$; $u.color = red$
  foreach $v \in \mathrm{Adj}[u]$ **do**





Für jeden Knoten u von G ist
- $u.d$ der Zeitpunkt der Entdeckung,
- $u.f$ der Abschluss-Zeitpunkt;

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

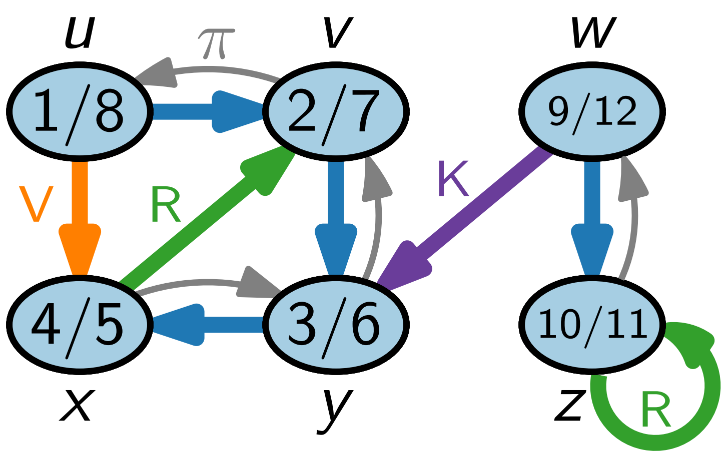
Tiefensuche – Pseudocode

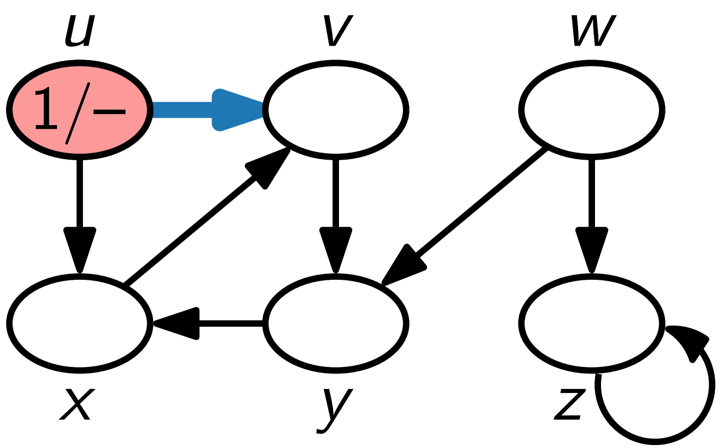
DFS(Graph G)
  foreach $u \in V(G)$ **do**
    $u.color = white$
    $u.\pi = nil$
  $time = 0$ globale Variable
  foreach $u \in V(G)$ **do**
    if $u.color == white$ **then** DFSVisit(G, u)

DFSVisit(Graph G, Vertex u)
  $time = time + 1$
  $u.d = time$; $u.color = red$
  foreach $v \in \mathrm{Adj}[u]$ **do**

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

$time = 1$





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)

foreach $u \in V(G)$ **do**

  $u.color = white$

  $u.\pi = nil$

$time = 0$ globale Variable

foreach $u \in V(G)$ **do**

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

DFSVisit(Graph G, Vertex u)

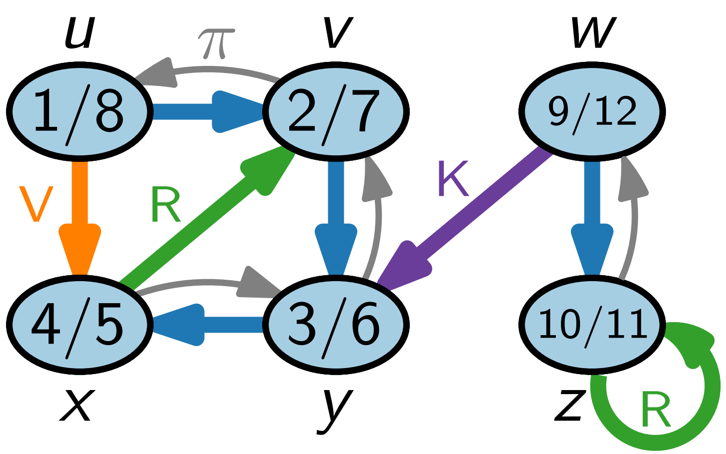
$time = time + 1$

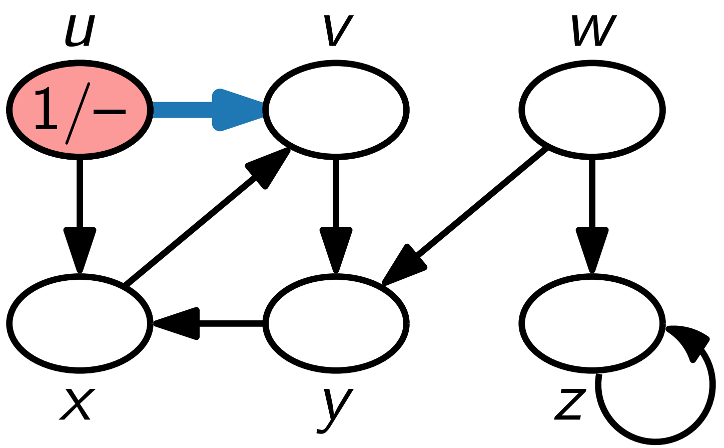
$u.d = time$; $u.color = red$

foreach $v \in \mathrm{Adj}[u]$ **do**

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

$time = 1$





Für jeden Knoten u von G ist
- $u.d$ der Zeitpunkt der Entdeckung,
- $u.f$ der Abschluss-Zeitpunkt;

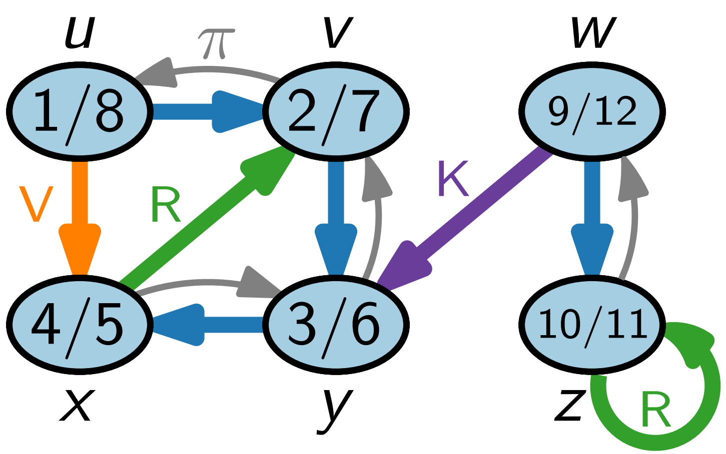
Besuchsintervall von u ist $[u.d, u.f]$.

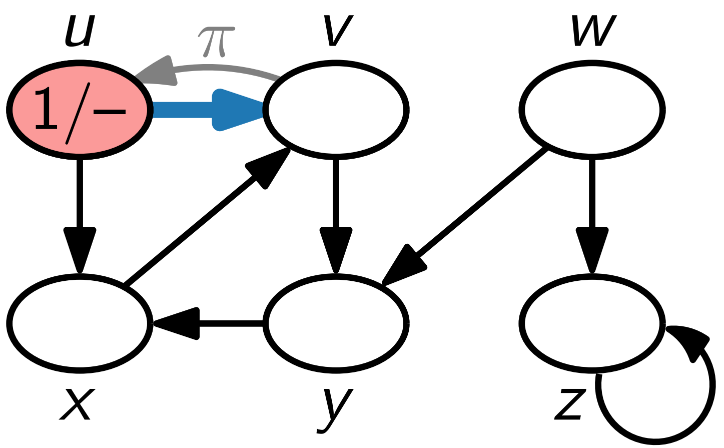
Tiefensuche – Pseudocode

DFS(Graph G)
foreach $u \in V(G)$ **do**
  $u.color = white$
  $u.\pi = nil$
$time = 0$ globale Variable
foreach $u \in V(G)$ **do**
  if $u.color == white$ **then** DFSVisit(G, u)

DFSVisit(Graph G, Vertex u)
$time = time + 1$
$u.d = time$; $u.color = red$
foreach $v \in \text{Adj}[u]$ **do**
  if $v.color == white$ **then**
    $v.\pi = u$;

$time = 1$





Für jeden Knoten u von G ist
- $u.d$ der Zeitpunkt der Entdeckung,
- $u.f$ der Abschluss-Zeitpunkt;

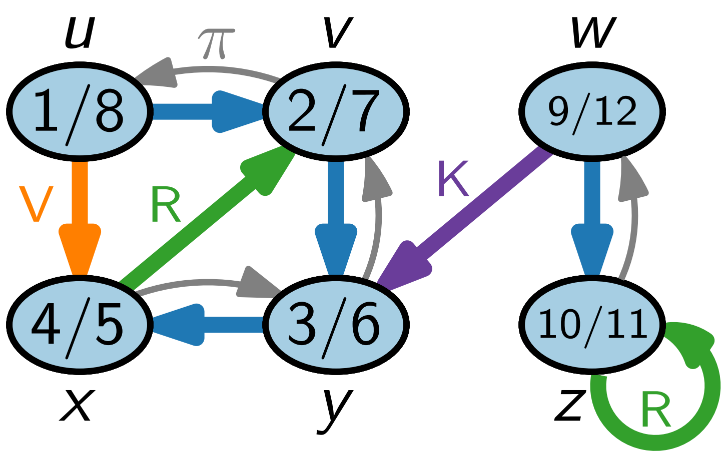
Besuchsintervall von u ist $[u.d, u.f]$.

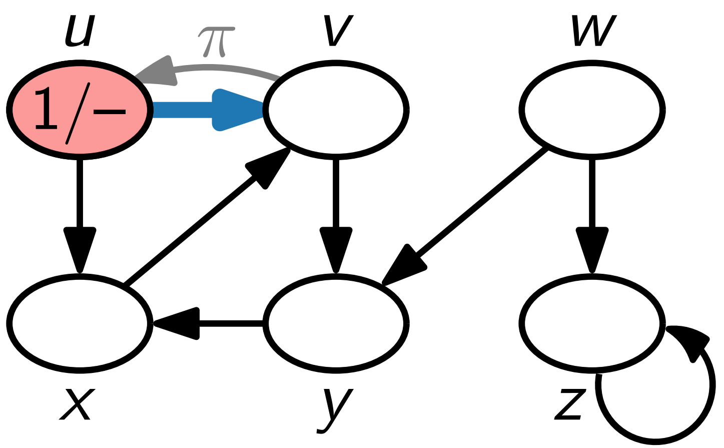
Tiefensuche – Pseudocode

DFS(Graph G)
  foreach $u \in V(G)$ **do**
    $u.color = white$
    $u.\pi = nil$
  $time = 0$ globale Variable
  foreach $u \in V(G)$ **do**
    if $u.color == white$ **then** DFSVisit(G, u)

DFSVisit(Graph G, Vertex u)
  $time = time + 1$
  $u.d = time$; $u.color = red$
  foreach $v \in \mathrm{Adj}[u]$ **do**
    if $v.color == white$ **then**
      $v.\pi = u$; DFSVisit(G, v)

$time = 1$





Für jeden Knoten u von G ist
- $u.d$ der Zeitpunkt der Entdeckung,
- $u.f$ der Abschluss-Zeitpunkt;

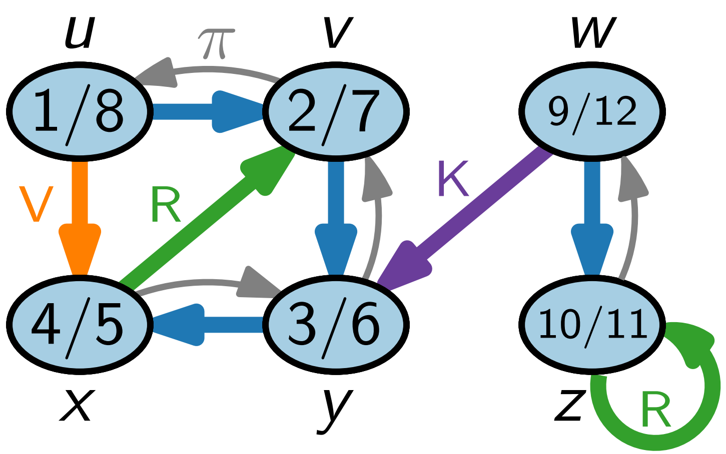
Besuchsintervall von u ist $[u.d, u.f]$.

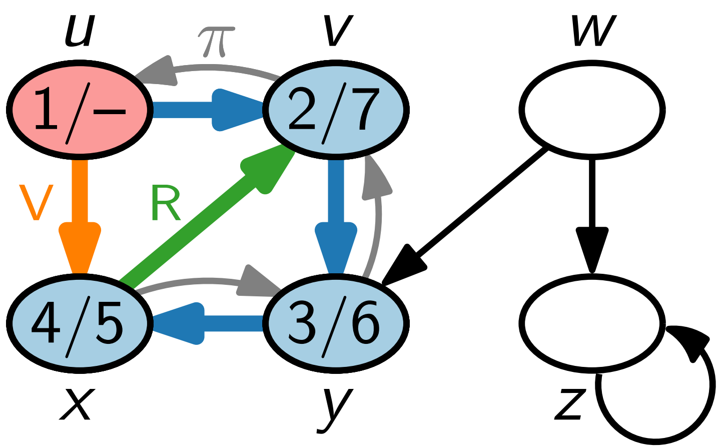
Tiefensuche – Pseudocode

```
DFS(Graph G)
  foreach u ∈ V(G) do
      u.color = white
      u.π = nil
  time = 0        globale Variable
  foreach u ∈ V(G) do
      if u.color == white then DFSVisit(G, u)
```

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

$time = 7$





Für jeden Knoten u von G ist
- $u.d$ der Zeitpunkt der Entdeckung,
- $u.f$ der Abschluss-Zeitpunkt;

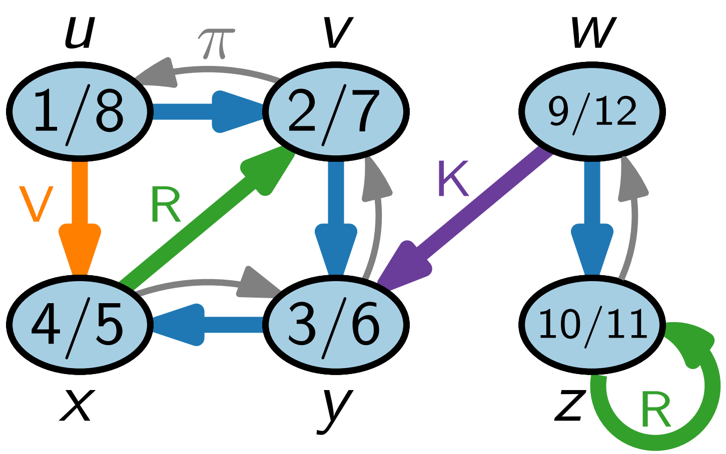
Besuchsintervall von u ist $[u.d, u.f]$.

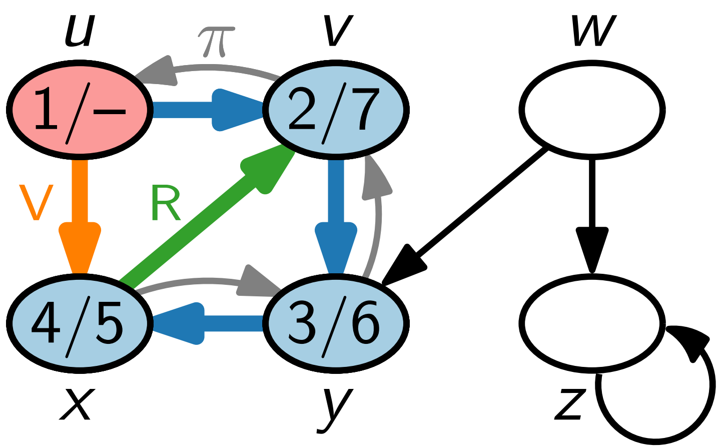
Tiefensuche – Pseudocode

DFS(Graph G)
  foreach $u \in V(G)$ **do**
    $u.color = white$
    $u.\pi = nil$
  $time = 0$ globale Variable
  foreach $u \in V(G)$ **do**
    if $u.color == white$ **then** DFSVisit(G, u)

DFSVisit(Graph G, Vertex u)
  $time = time + 1$
  $u.d = time$; $u.color = red$
  foreach $v \in \mathrm{Adj}[u]$ **do**
    if $v.color == white$ **then**
      $v.\pi = u$; DFSVisit(G, v)
  $time = time + 1$

$time = 8$





Für jeden Knoten u von G ist
- $u.d$ der Zeitpunkt der Entdeckung,
- $u.f$ der Abschluss-Zeitpunkt;

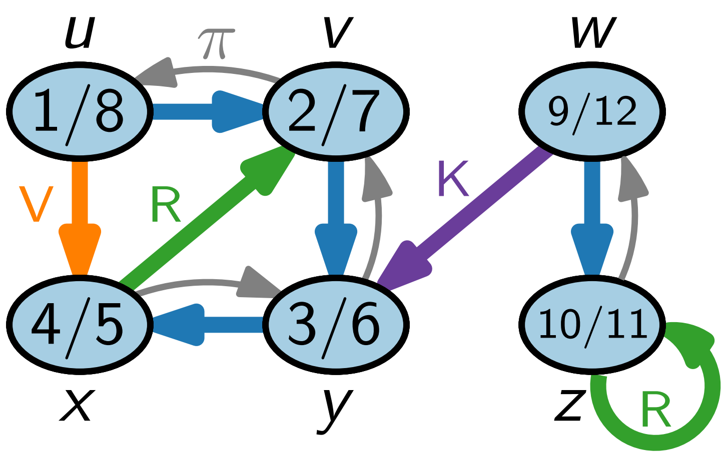
Besuchsintervall von u ist $[u.d, u.f]$.

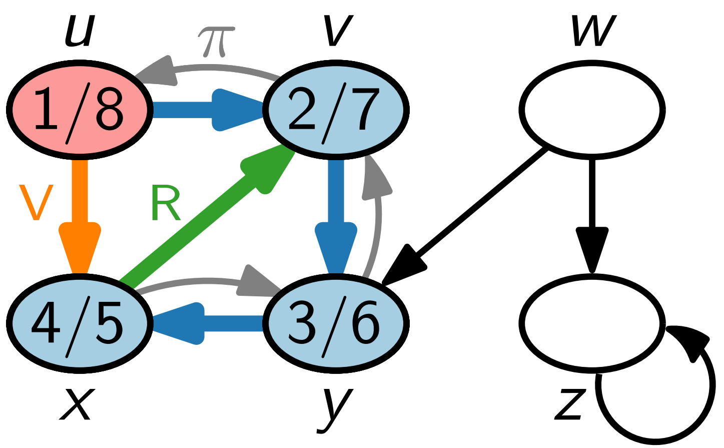
Tiefensuche – Pseudocode

DFS(Graph G)
  foreach $u \in V(G)$ **do**
    $u.color = white$
    $u.\pi = nil$
  $time = 0$ globale Variable
  foreach $u \in V(G)$ **do**
    if $u.color == white$ **then** DFSVisit(G, u)

DFSVisit(Graph G, Vertex u)
  $time = time + 1$
  $u.d = time$; $u.color = red$
  foreach $v \in \mathrm{Adj}[u]$ **do**
    if $v.color == white$ **then**
      $v.\pi = u$; DFSVisit(G, v)
  $time = time + 1$
  $u.f = time$;

$time = 8$





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

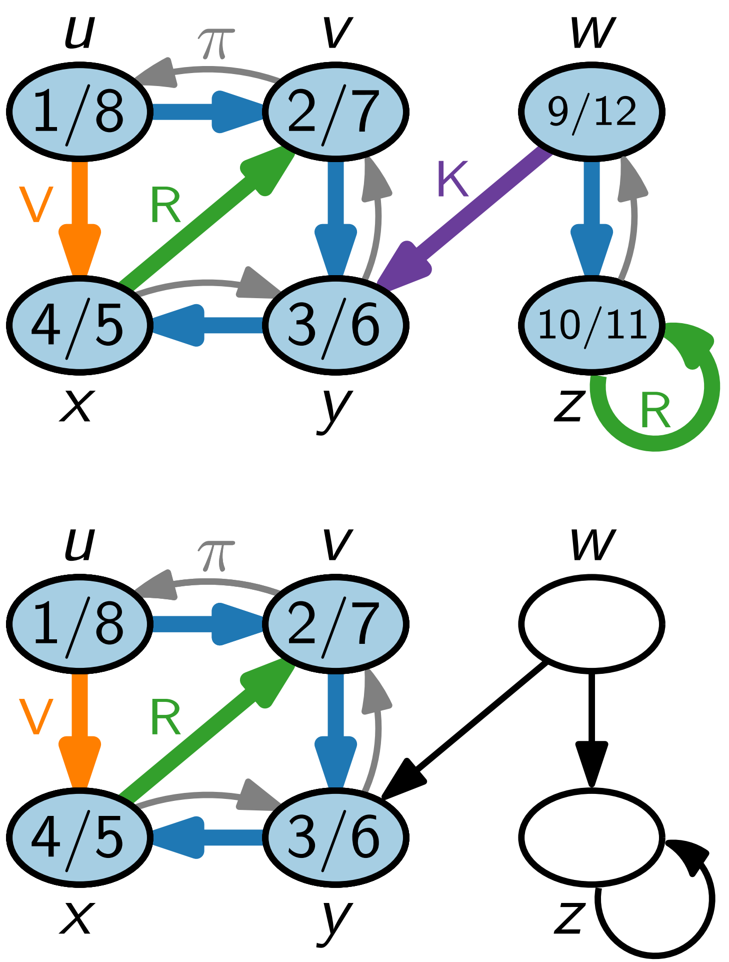
$\mathrm{DFS}(\text{Graph } G)$

foreach $u \in V(G)$ **do**
  $u.\mathit{color} = \mathit{white}$
  $u.\pi = \mathit{nil}$
$\mathit{time} = 0$ globale Variable
foreach $u \in V(G)$ **do**
  if $u.\mathit{color} == \mathit{white}$ **then** $\mathrm{DFSVisit}(G, u)$

$\mathrm{DFSVisit}(\text{Graph } G, \text{Vertex } u)$

$\mathit{time} = \mathit{time} + 1$
$u.d = \mathit{time}$; $u.\mathit{color} = \mathit{red}$
foreach $v \in \mathrm{Adj}[u]$ **do**
  if $v.\mathit{color} == \mathit{white}$ **then**
    $v.\pi = u$; $\mathrm{DFSVisit}(G, v)$
$\mathit{time} = \mathit{time} + 1$
$u.f = \mathit{time}$; $u.\mathit{color} = \mathit{blue}$

$\mathit{time} = 8$



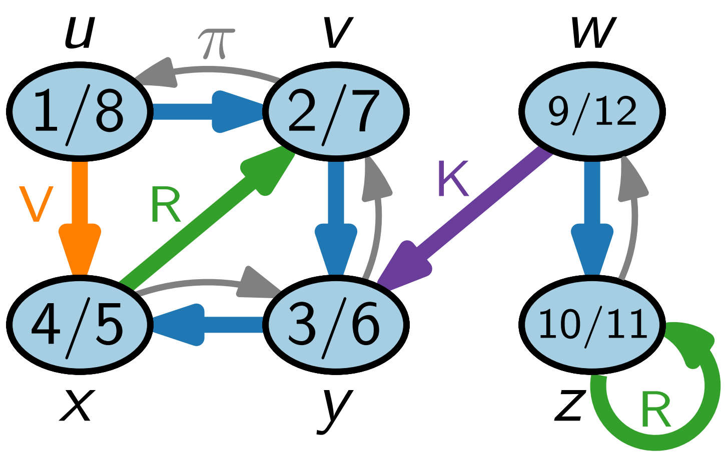
Für jeden Knoten u von G ist
- $u.d$ der Zeitpunkt der Entdeckung,
- $u.f$ der Abschluss-Zeitpunkt;

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

Tiefensuche – Pseudocode

DFS(Graph G)
  foreach $u \in V(G)$ **do**
    $u.color = white$
    $u.\pi = nil$
  $time = 0$ globale Variable
  foreach $u \in V(G)$ **do**
    if $u.color == white$ **then** DFSVisit(G, u)

DFSVisit(Graph G, Vertex u)
  $time = time + 1$
  $u.d = time$; $u.color = red$
  foreach $v \in \mathrm{Adj}[u]$ **do**
    if $v.color == white$ **then**
      $v.\pi = u$; DFSVisit(G, v)
  $time = time + 1$
  $u.f = time$; $u.color = blue$

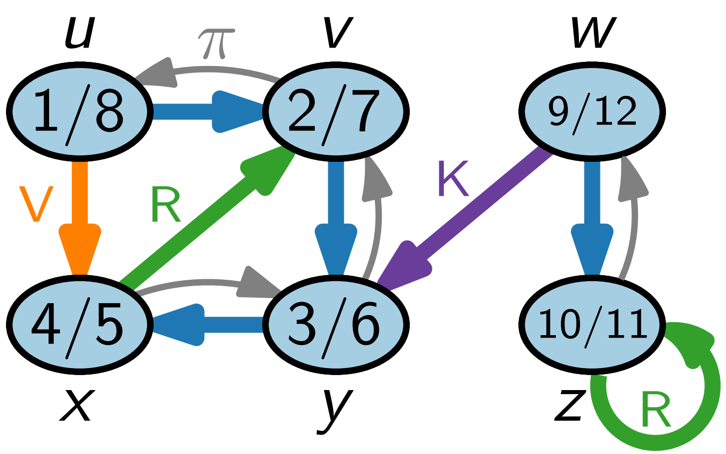


Laufzeit von DFS?

Tiefensuche – Pseudocode

DFS(Graph G)
  foreach $u \in V(G)$ **do**
    $u.color = white$
    $u.\pi = nil$
  $time = 0$ globale Variable
  foreach $u \in V(G)$ **do**
    if $u.color == white$ **then** DFSVisit(G, u)

DFSVisit(Graph G, Vertex u)
  $time = time + 1$
  $u.d = time$; $u.color = red$
  foreach $v \in \text{Adj}[u]$ **do**
    if $v.color == white$ **then**
      $v.\pi = u$; DFSVisit(G, v)
  $time = time + 1$
  $u.f = time$; $u.color = blue$

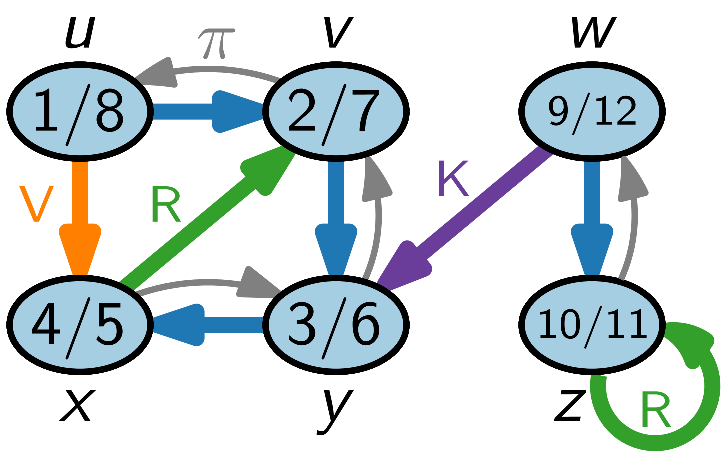


Laufzeit von DFS?

- DFSVisit wird nur für weiße Knoten aufgerufen.

Tiefensuche – Pseudocode

```
DFS(Graph G)
  foreach u ∈ V(G) do
     u.color = white
     u.π = nil
  time = 0        globale Variable
  foreach u ∈ V(G) do
     if u.color == white then DFSVisit(G, u)
```



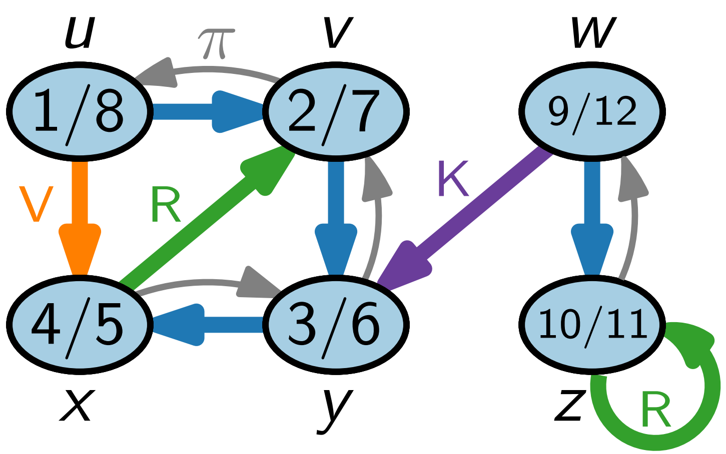
Laufzeit von DFS?

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

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

Tiefensuche – Pseudocode

```
DFS(Graph G)
  foreach u ∈ V(G) do
  │   u.color = white
  └   u.π = nil
  time = 0          globale Variable
  foreach u ∈ V(G) 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 = red
  foreach v ∈ Adj[u] do
  │   if v.color == white then
  └       └ v.π = u; DFSVisit(G, v)
  time = time + 1
  u.f = time; u.color = blue
```

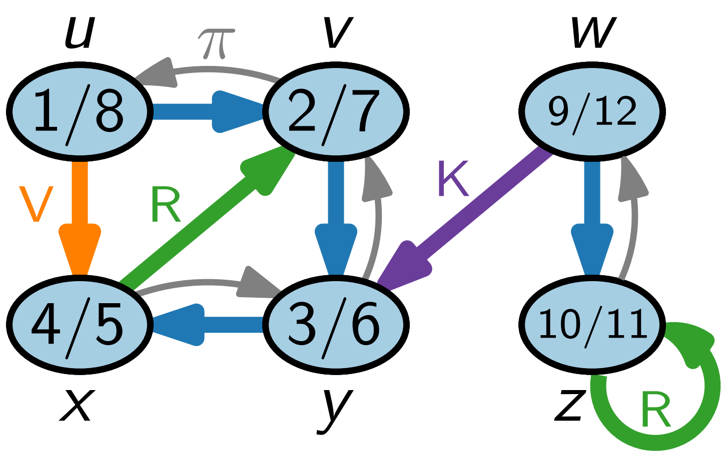
- DFSVisit wird nur für weiße Knoten aufgerufen.
- In DFSVisit wird der neue Knoten sofort rot gefärbt.

$\Rightarrow$ DFSVisit wird für jeden Knoten genau $1\times$ aufgerufen.

Tiefensuche – Pseudocode

DFS(Graph G)
  foreach $u \in V(G)$ **do**
    $u.color = white$
    $u.\pi = nil$
  $time = 0$ globale Variable
  foreach $u \in V(G)$ **do**
    if $u.color == white$ **then** DFSVisit(G, u)

DFSVisit(Graph G, Vertex u)
  $time = time + 1$
  $u.d = time$; $u.color = red$
  foreach $v \in Adj[u]$ **do**
    if $v.color == white$ **then**
      $v.\pi = u$; DFSVisit(G, v)
  $time = time + 1$
  $u.f = time$; $u.color = blue$



Laufzeit von DFS?

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

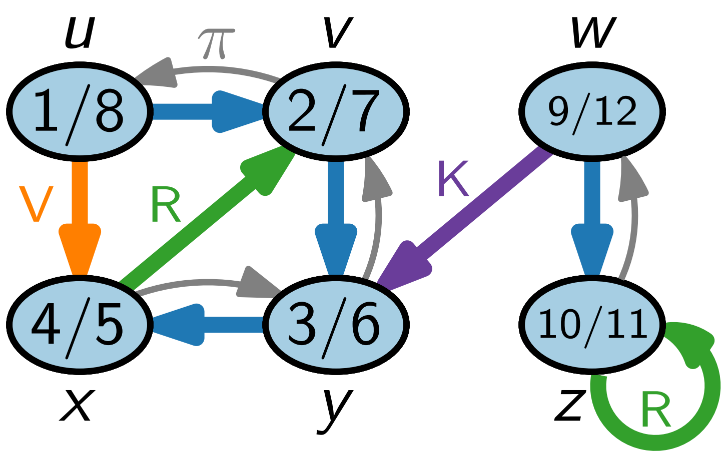
$\Rightarrow$ DFSVisit wird für jeden Knoten genau $1\times$ aufgerufen.

- DFS ohne **if** $\mathcal{O}(V)$ Zeit
  DFSVisit ohne Rek. $\mathcal{O}((\text{out})\deg(u))$

Tiefensuche – Pseudocode

DFS(Graph G)
foreach $u \in V(G)$ **do**
  $u.color = white$
  $u.\pi = nil$
$time = 0$ globale Variable
foreach $u \in V(G)$ **do**
  if $u.color == white$ **then** DFSVisit(G, u)

DFSVisit(Graph G, Vertex u)
$time = time + 1$
$u.d = time$; $u.color = red$
foreach $v \in \mathrm{Adj}[u]$ **do**
  if $v.color == white$ **then**
    $v.\pi = u$; DFSVisit(G, v)
$time = time + 1$
$u.f = time$; $u.color = blue$



Laufzeit von DFS?

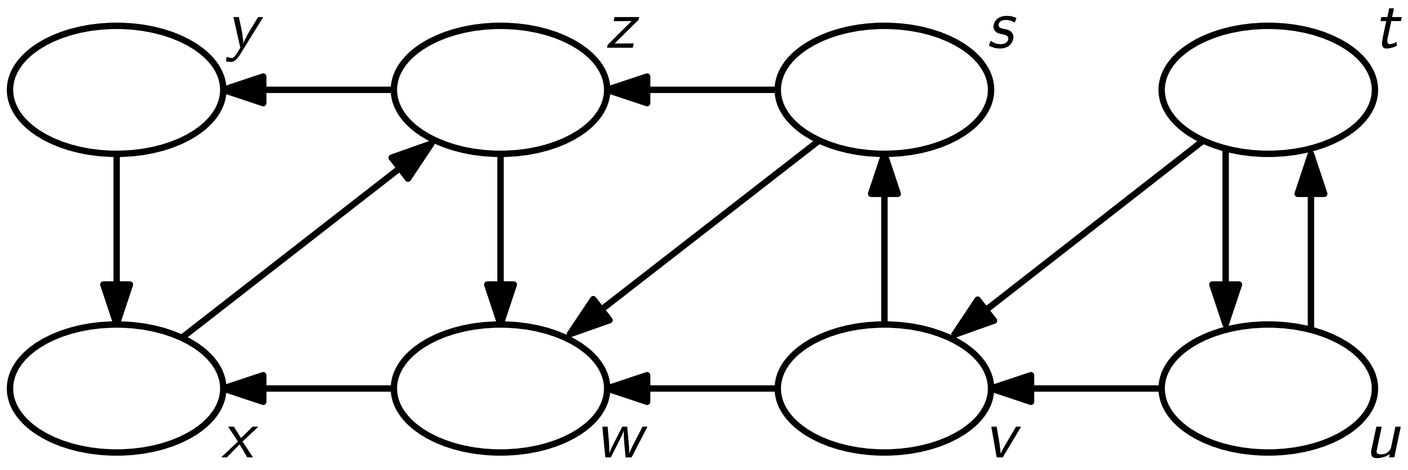
- DFSVisit wird nur für weiße Knoten aufgerufen.
- In DFSVisit wird der neue Knoten sofort rot gefärbt.

$\Rightarrow$ DFSVisit wird für jeden Knoten genau $1\times$ aufgerufen.

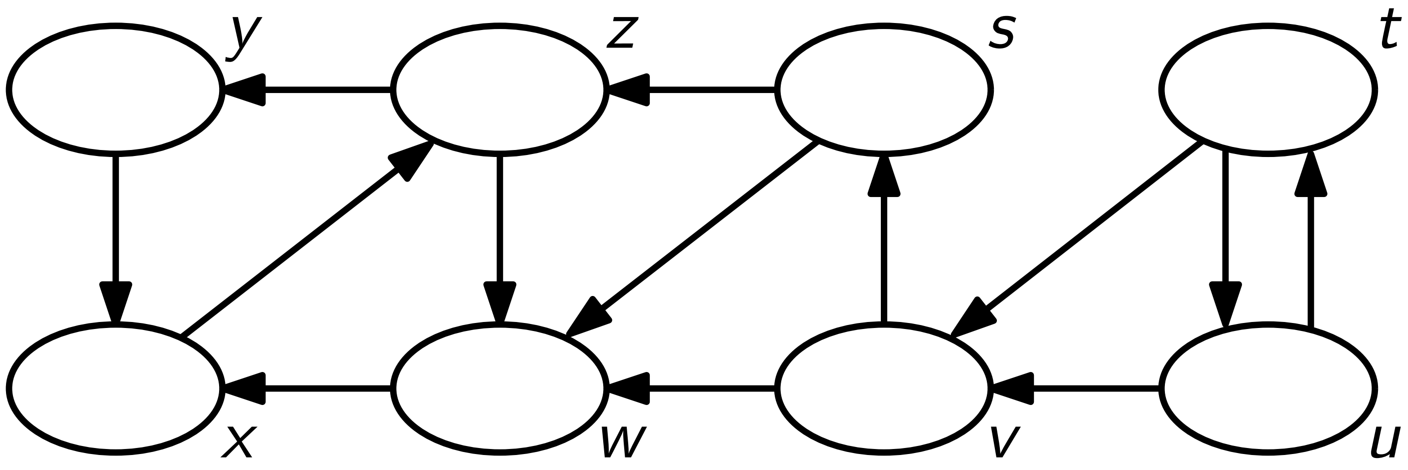
- DFS ohne **if** — $\mathcal{O}(V)$ Zeit
  DFSVisit ohne Rek. — $\mathcal{O}((\mathrm{out})\deg(u))$

DFS gesamt — $\mathcal{O}(V + E)$ Zeit

Tiefensuche – Eigenschaften



Tiefensuche – Eigenschaften

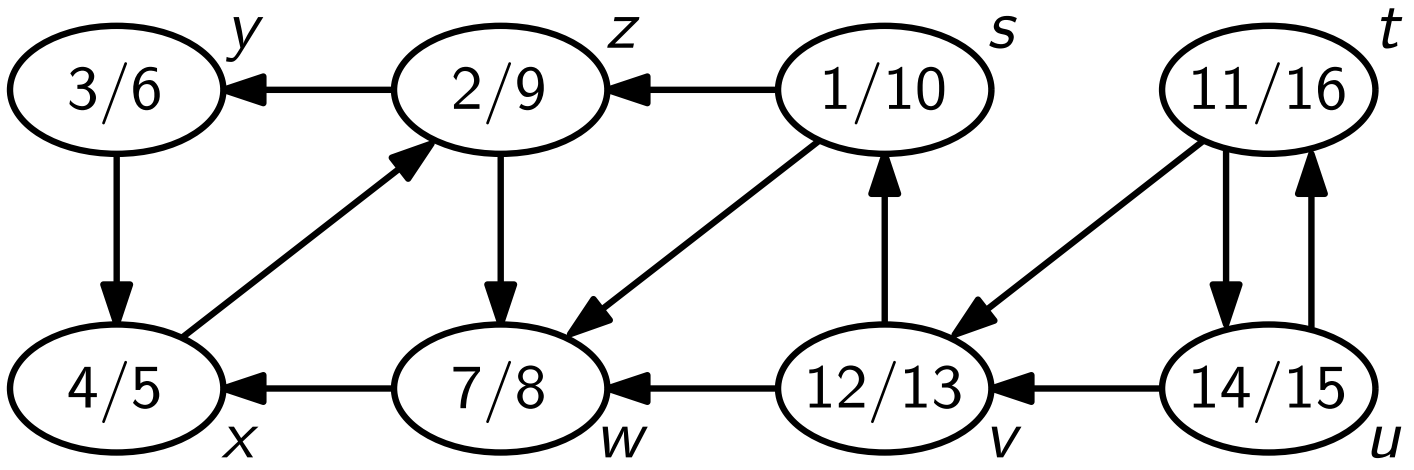


Aufgabe: Kopieren Sie obigen Graphen.

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

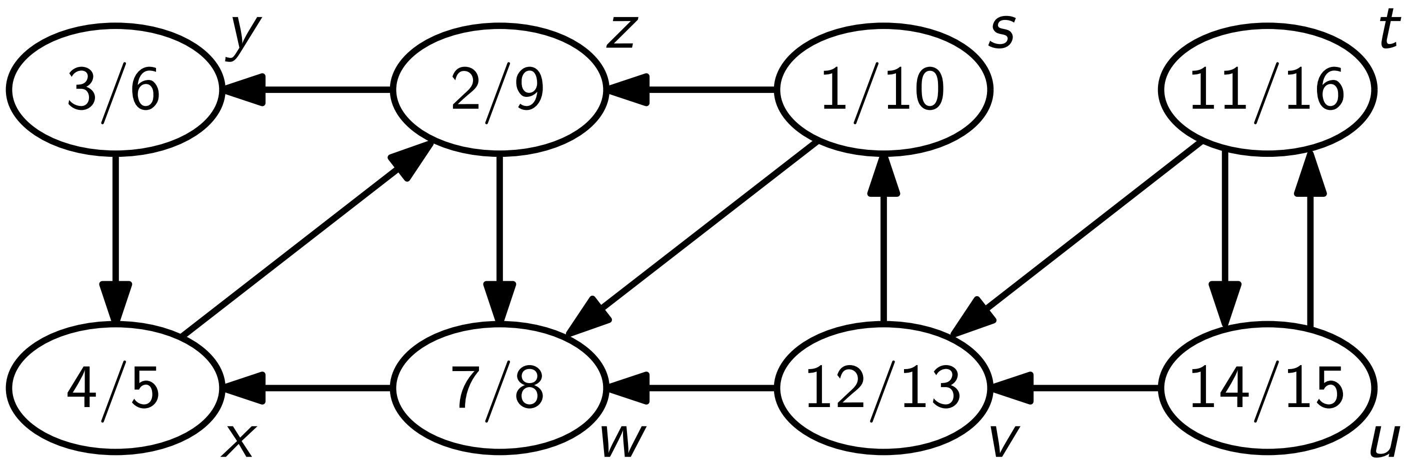


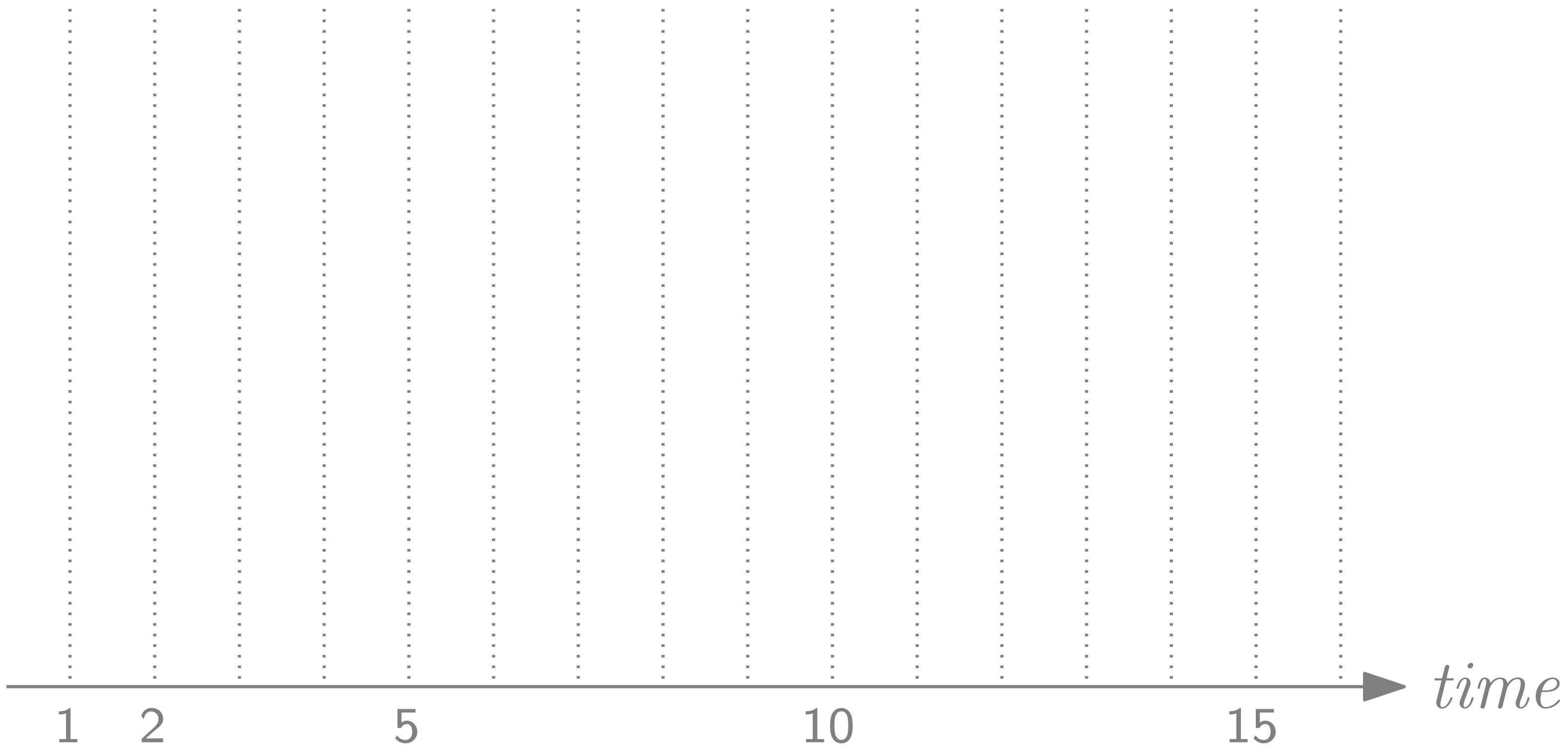
Aufgabe: Kopieren Sie obigen Graphen.

Berechnen Sie dann mit DFS alle Besuchsintervalle.

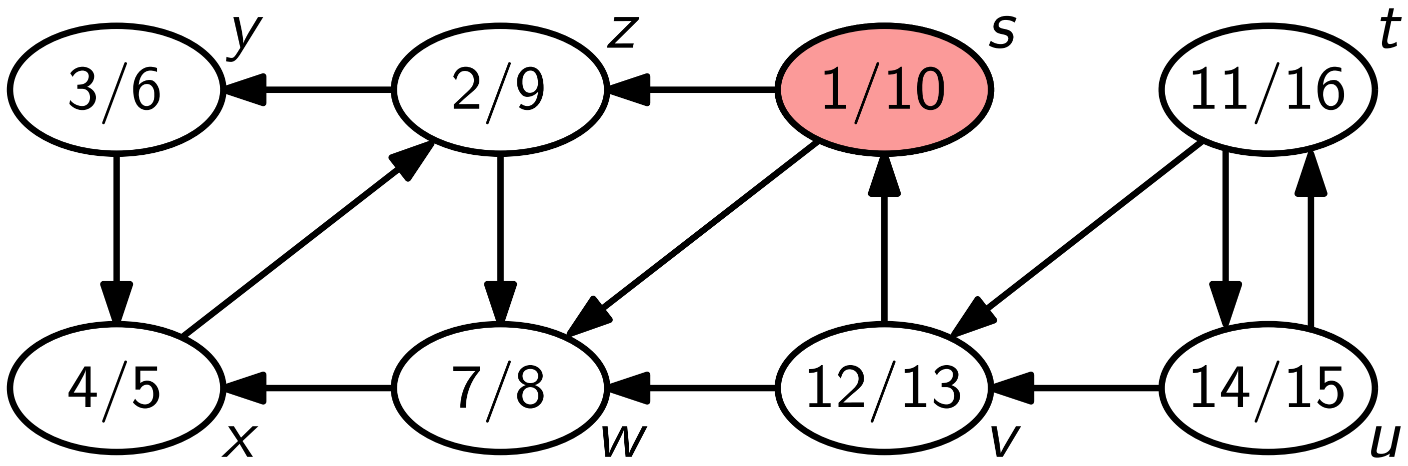
Beginnen Sie mit s. Wenn Sie eine Wahl haben, nehmen Sie zuerst den **obersten** verfügbaren Knoten.

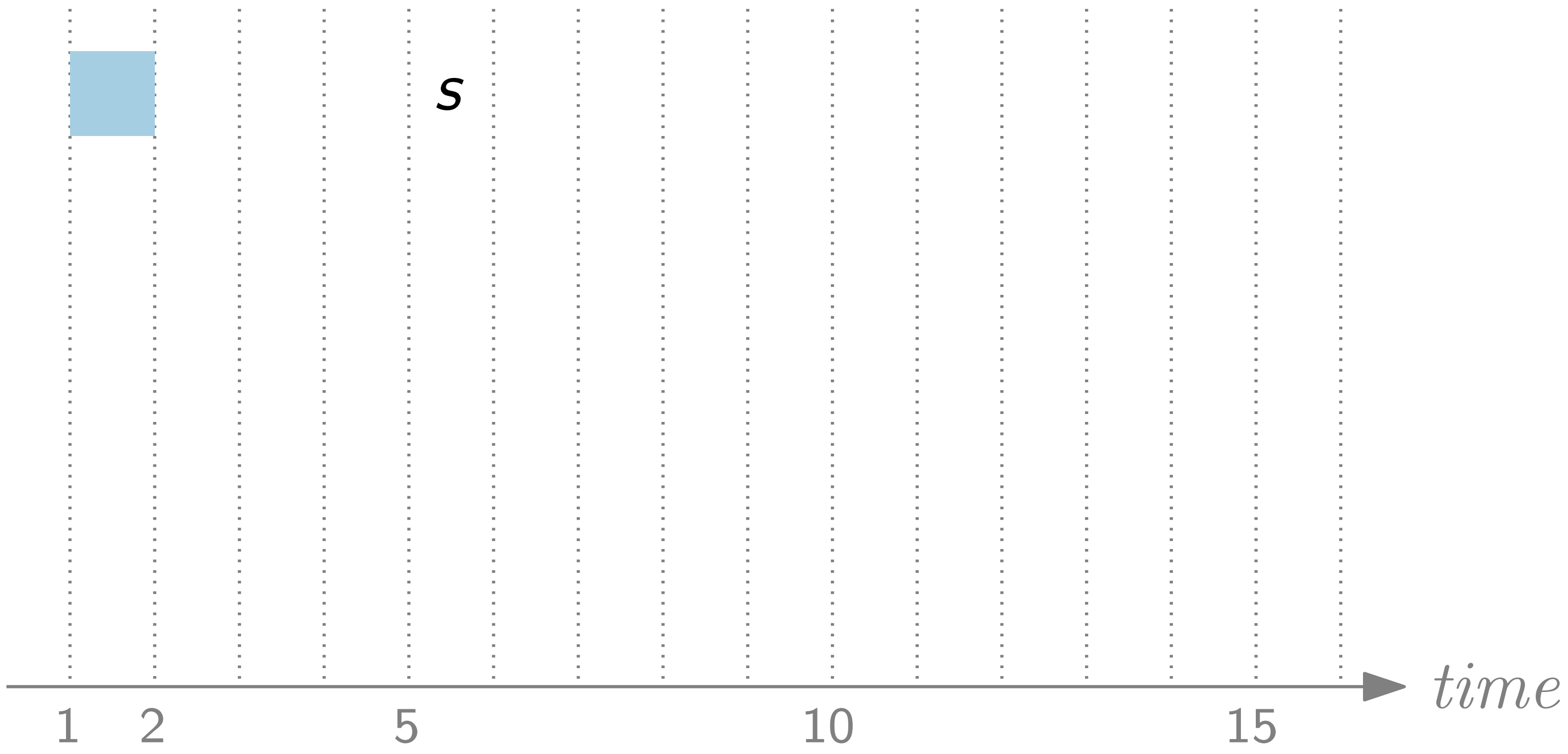
Tiefensuche – Eigenschaften



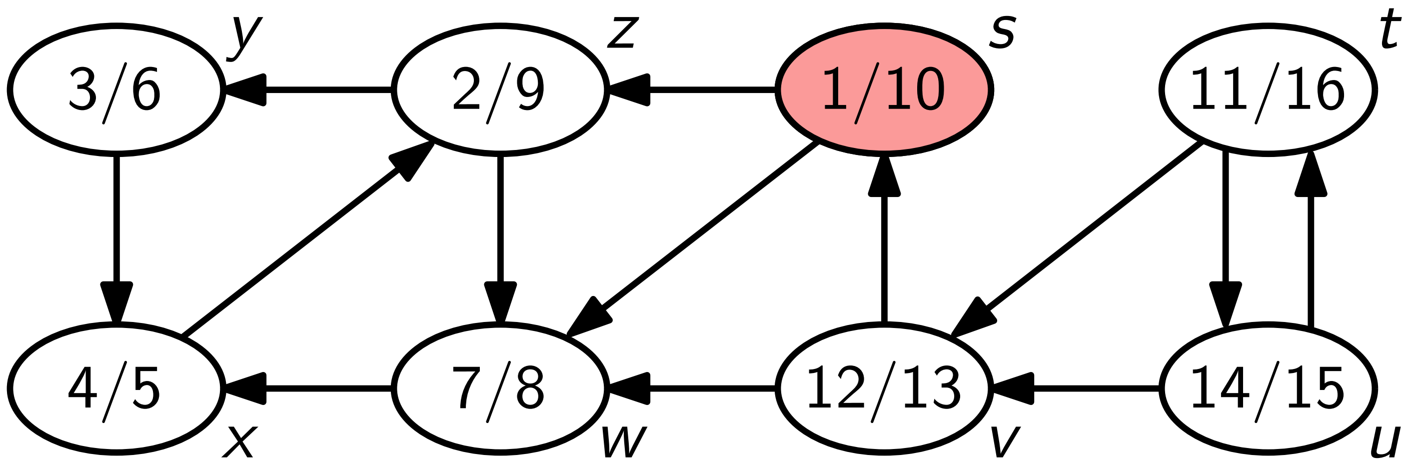


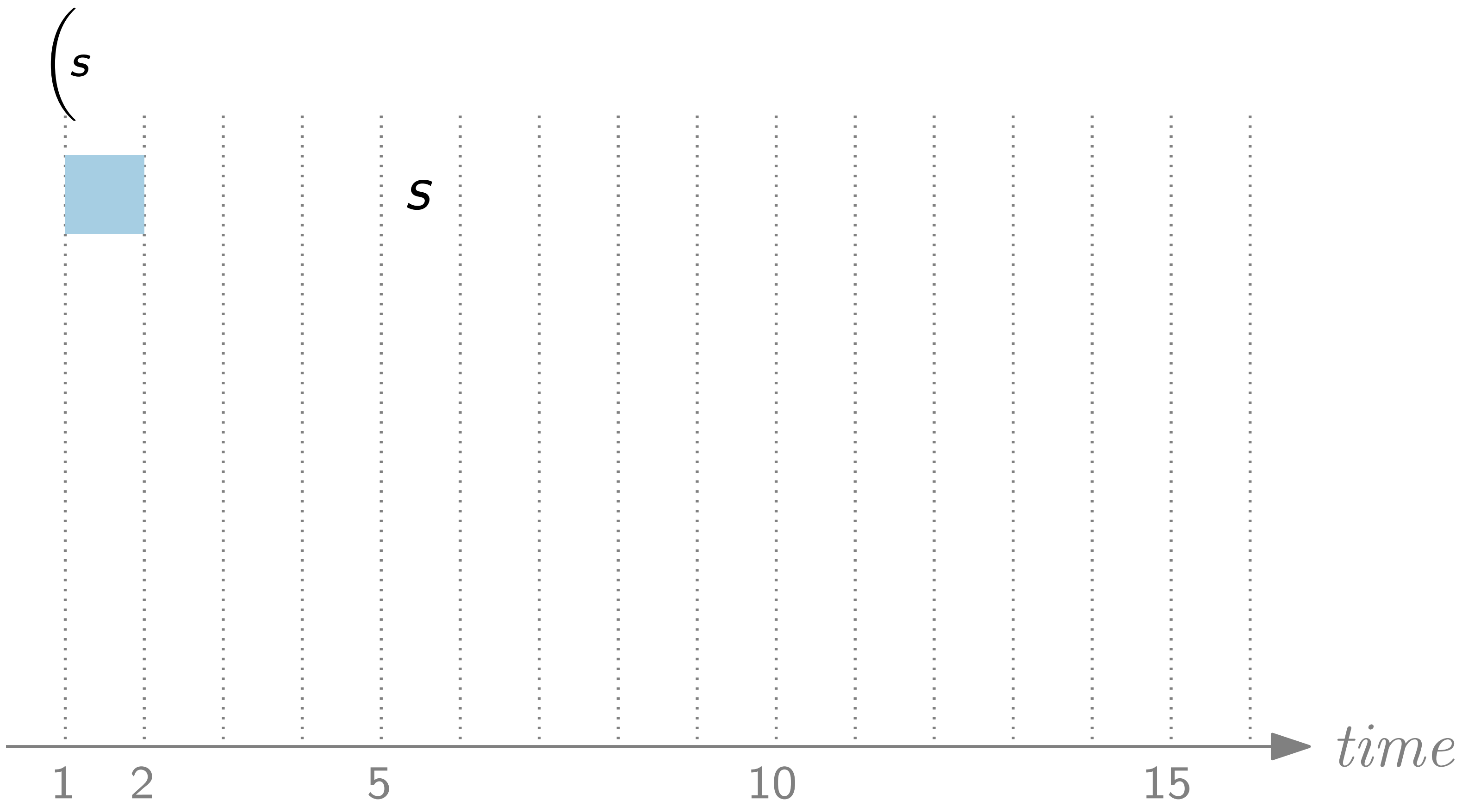
Tiefensuche – Eigenschaften



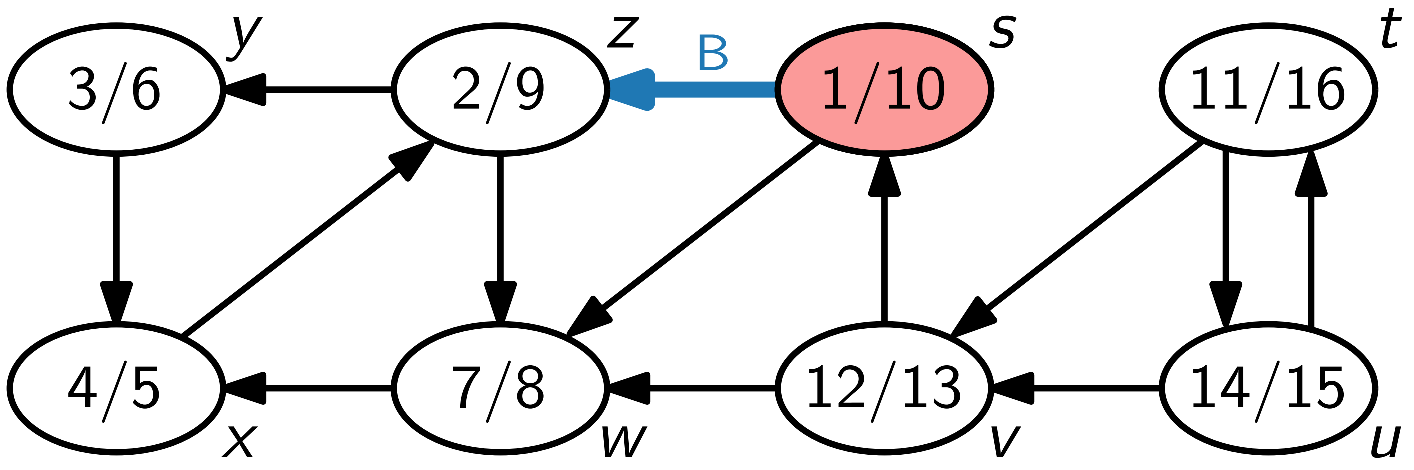


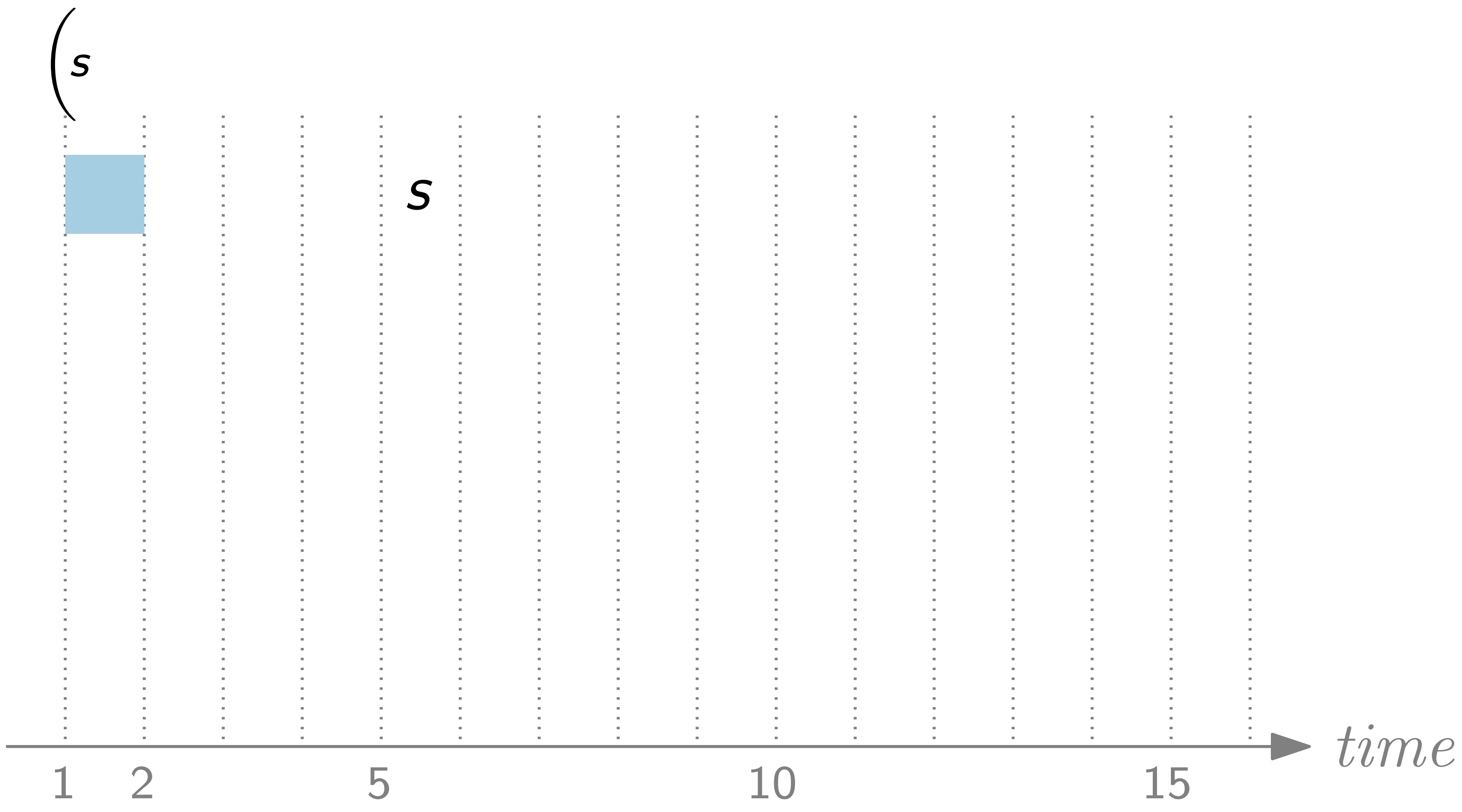
Tiefensuche – Eigenschaften



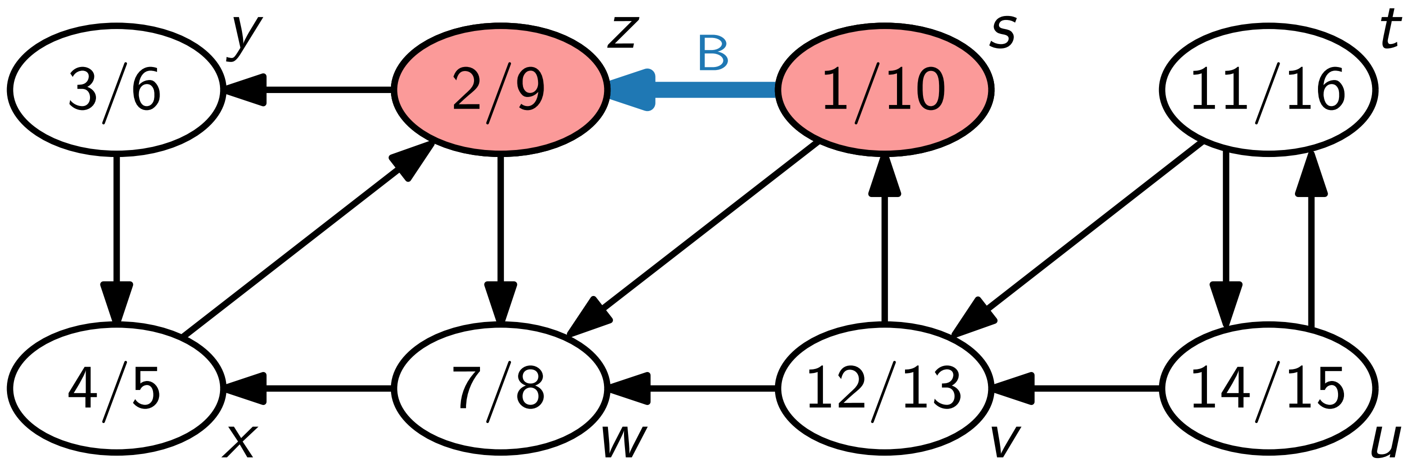


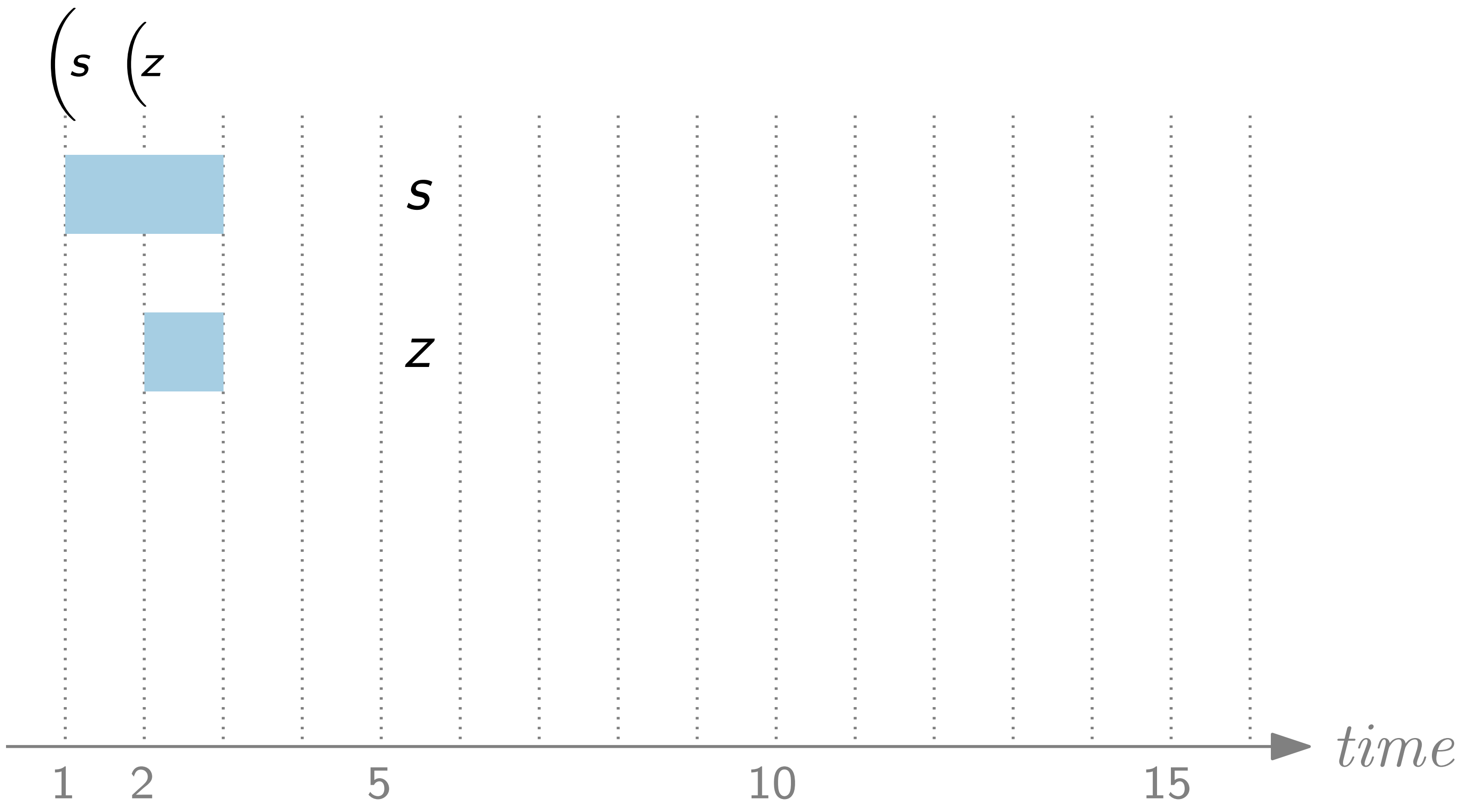
Tiefensuche – Eigenschaften



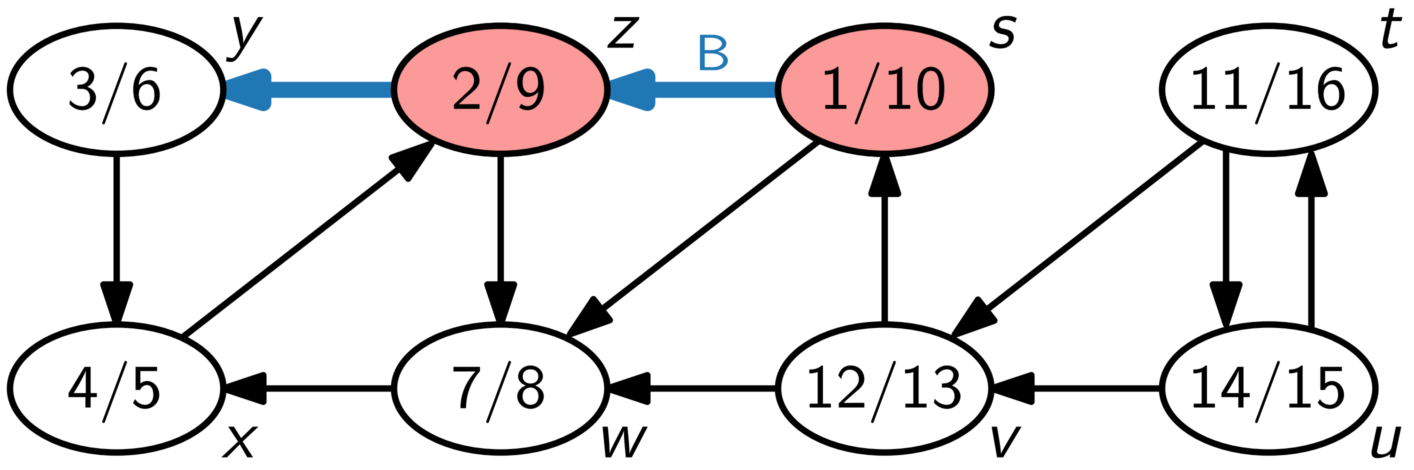


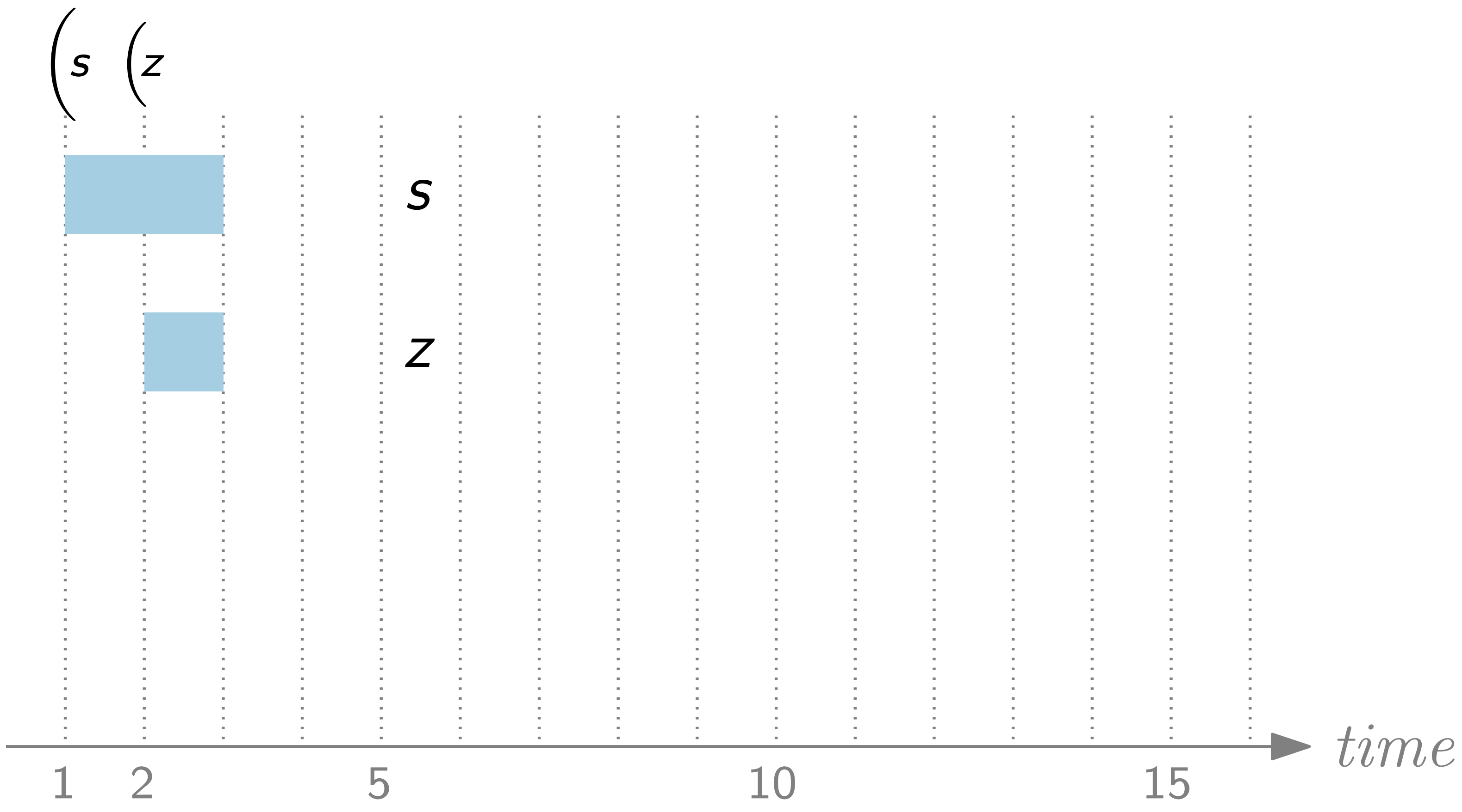
Tiefensuche – Eigenschaften



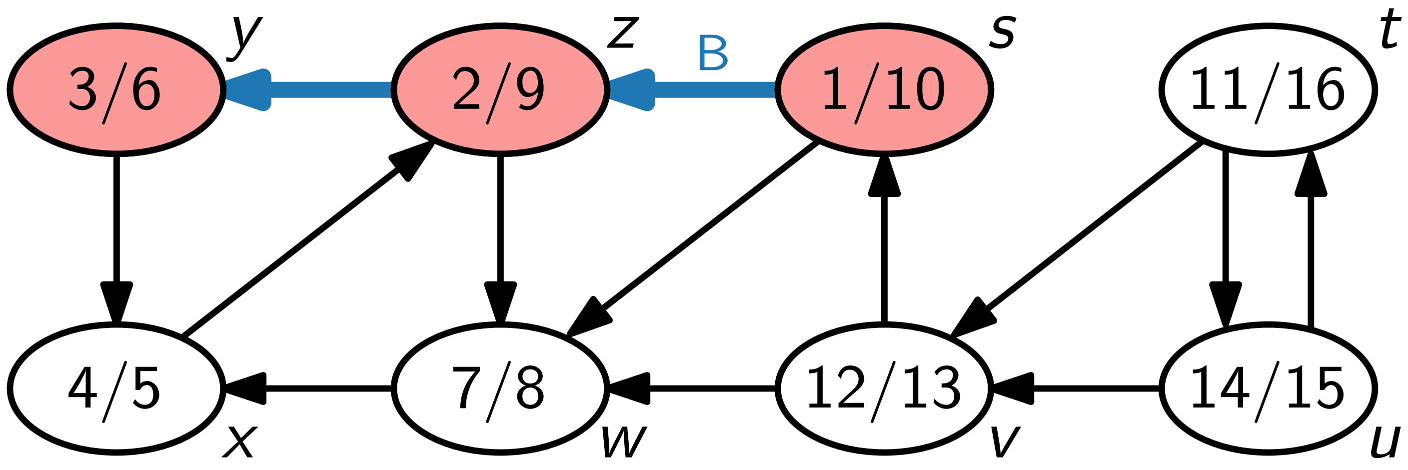


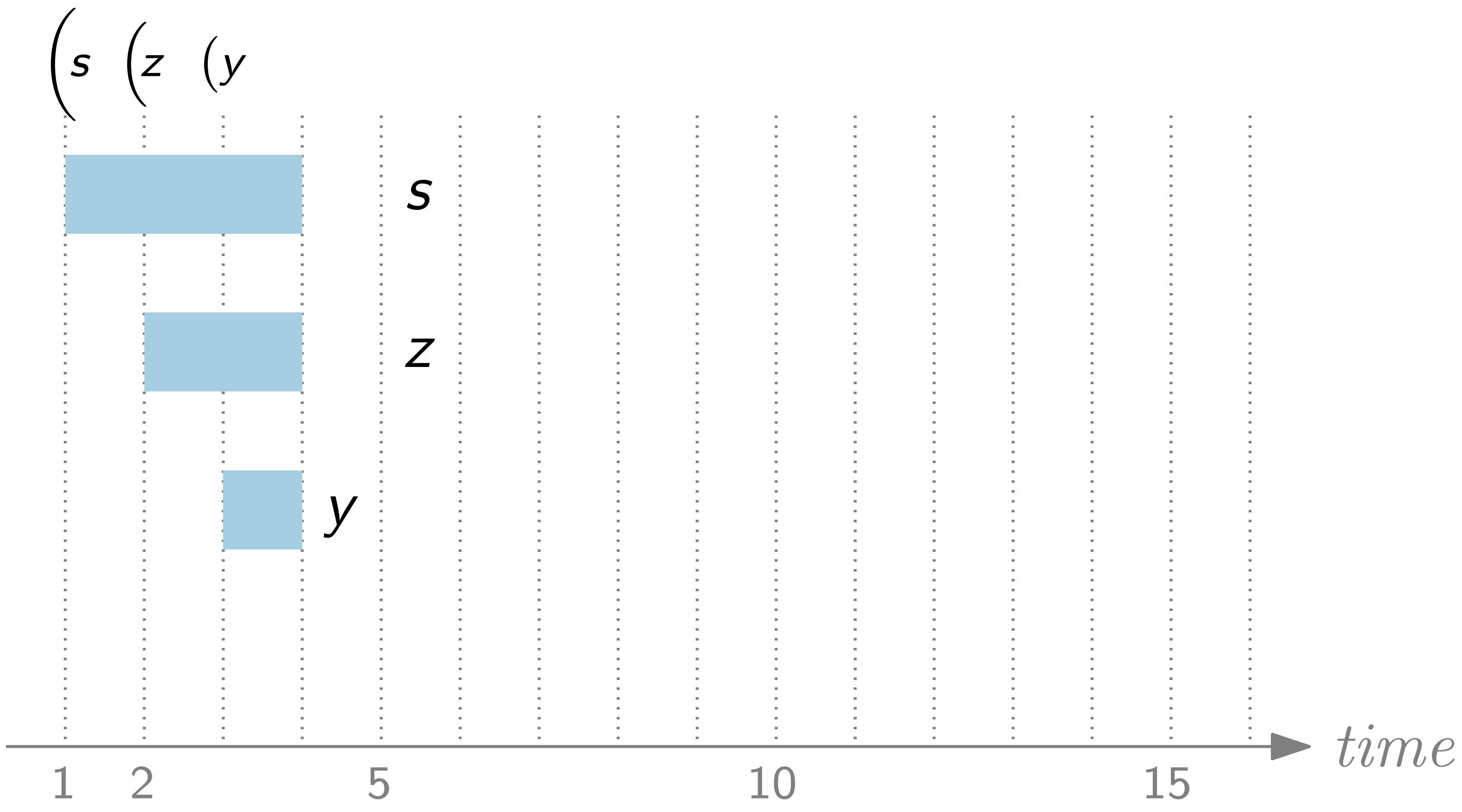
Tiefensuche – Eigenschaften



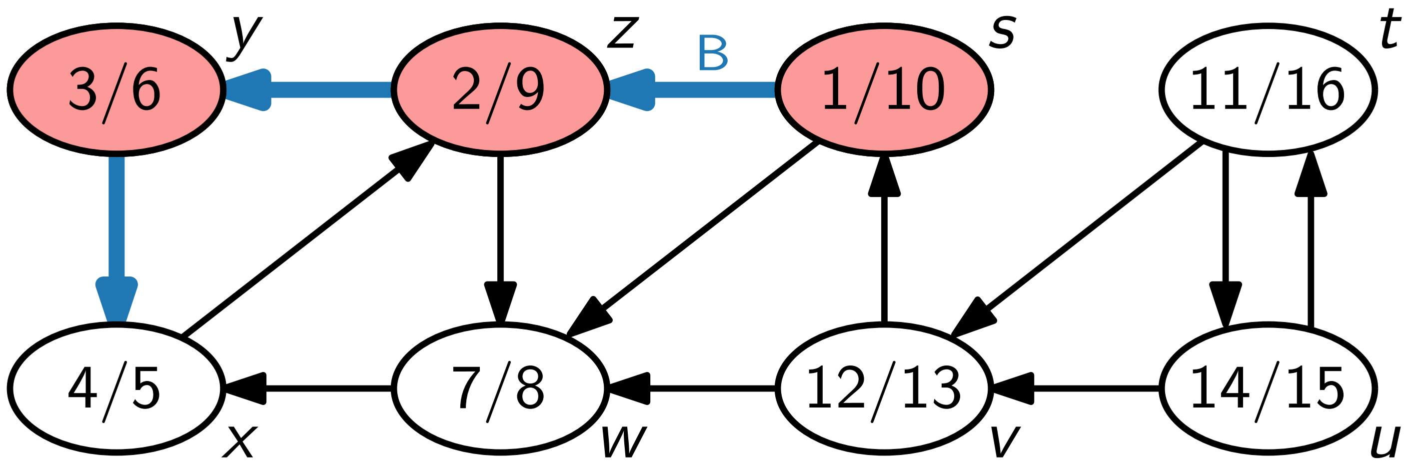


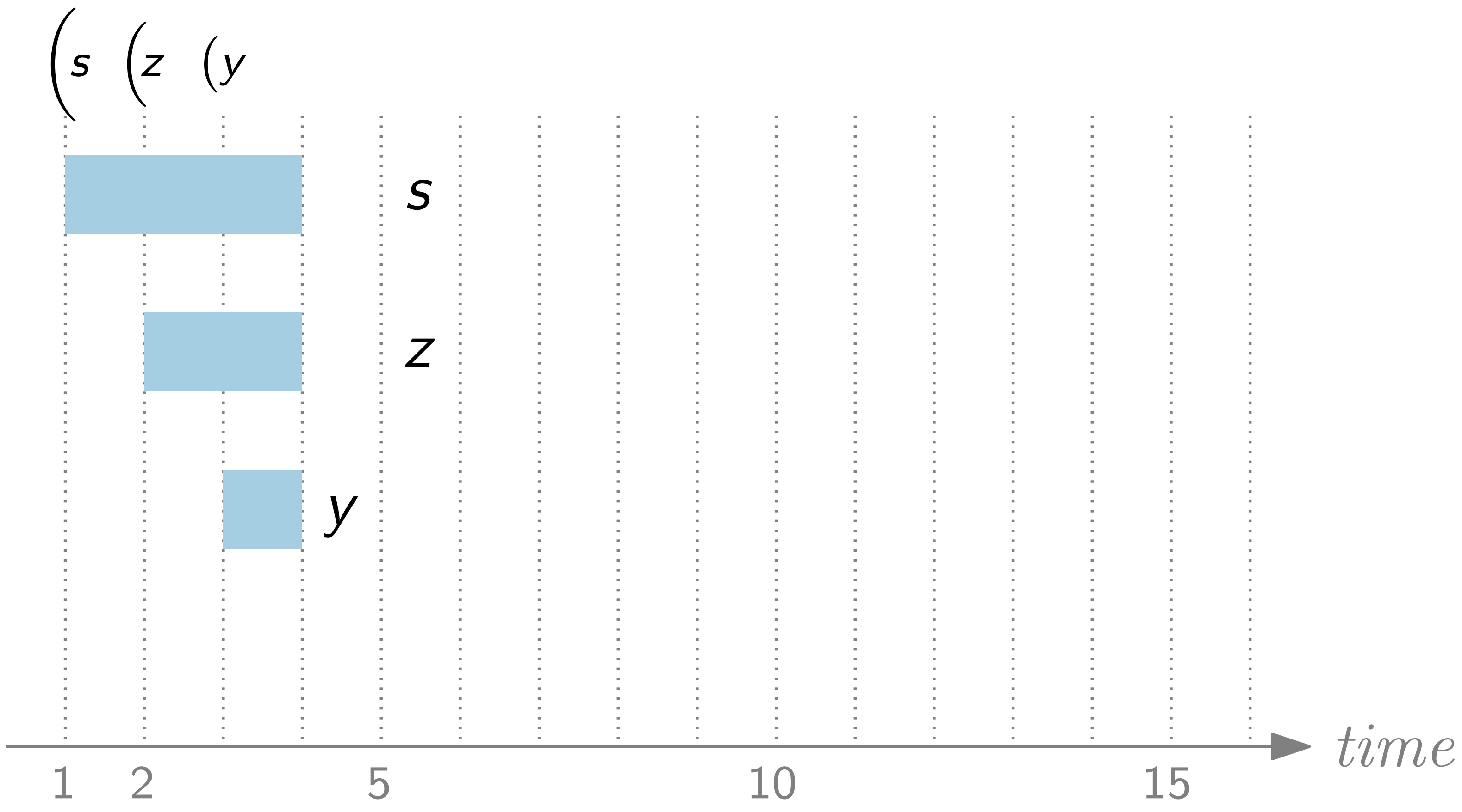
Tiefensuche – Eigenschaften



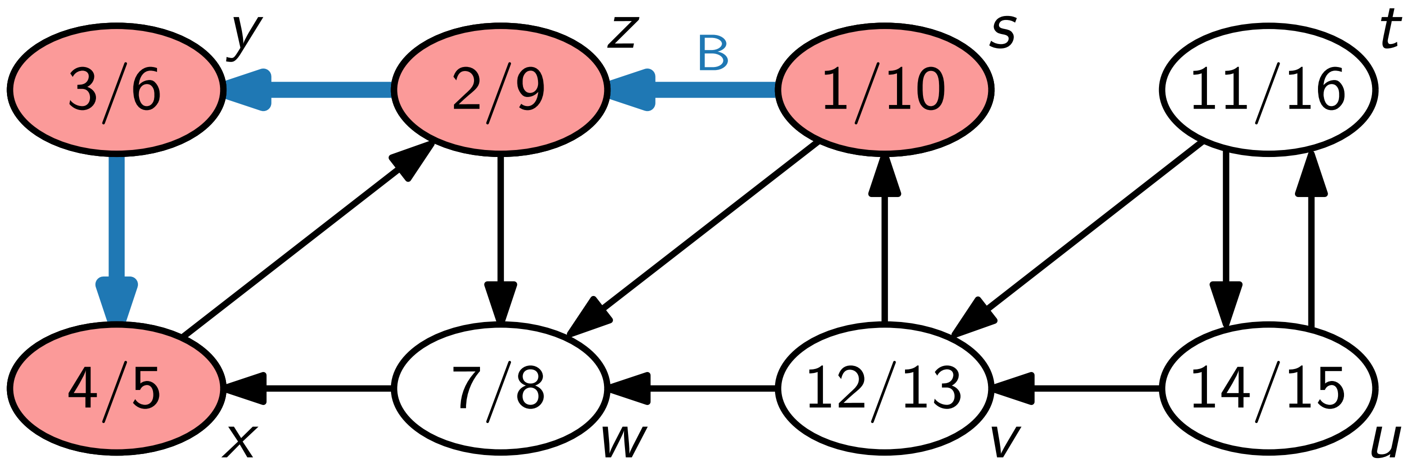


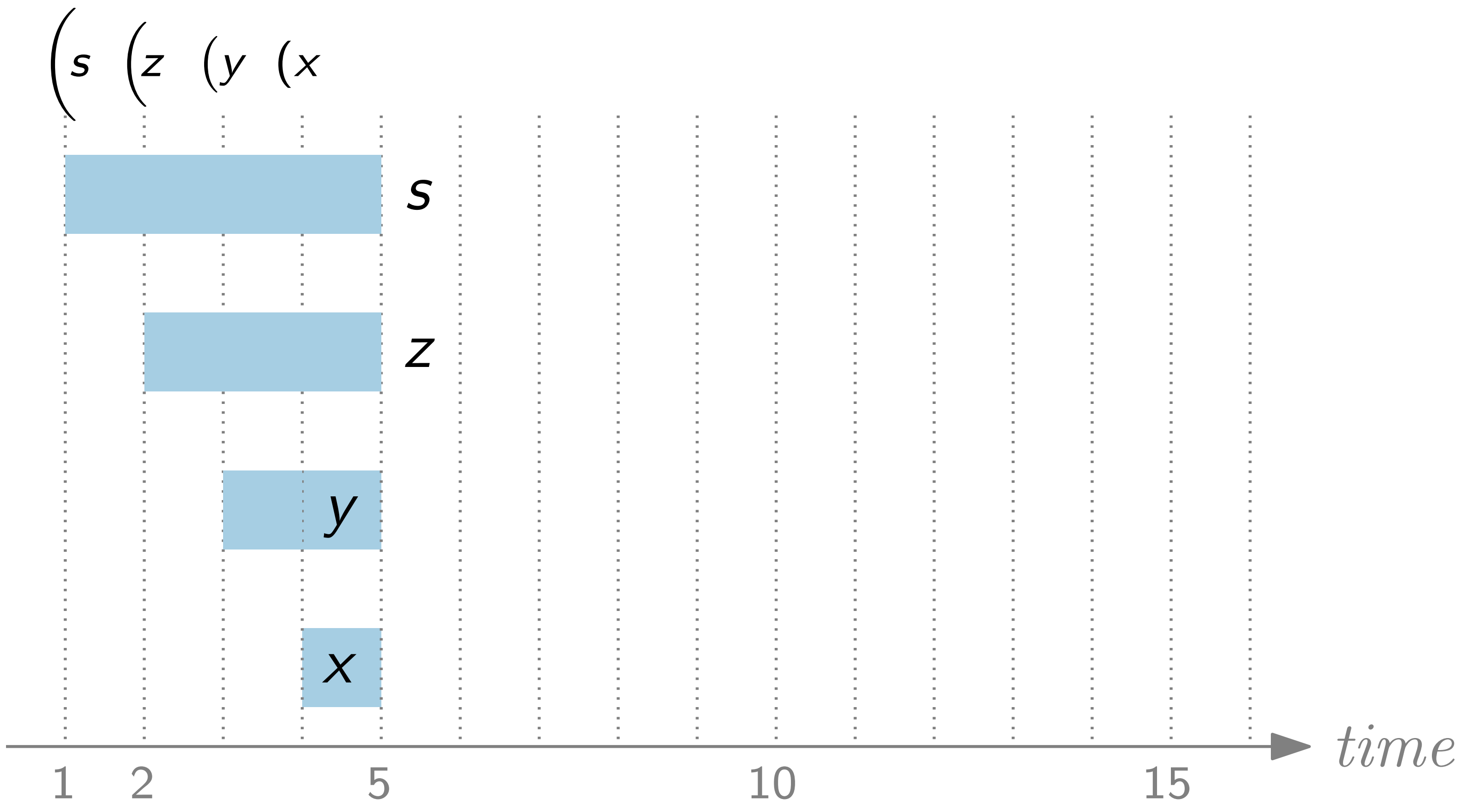
Tiefensuche – Eigenschaften



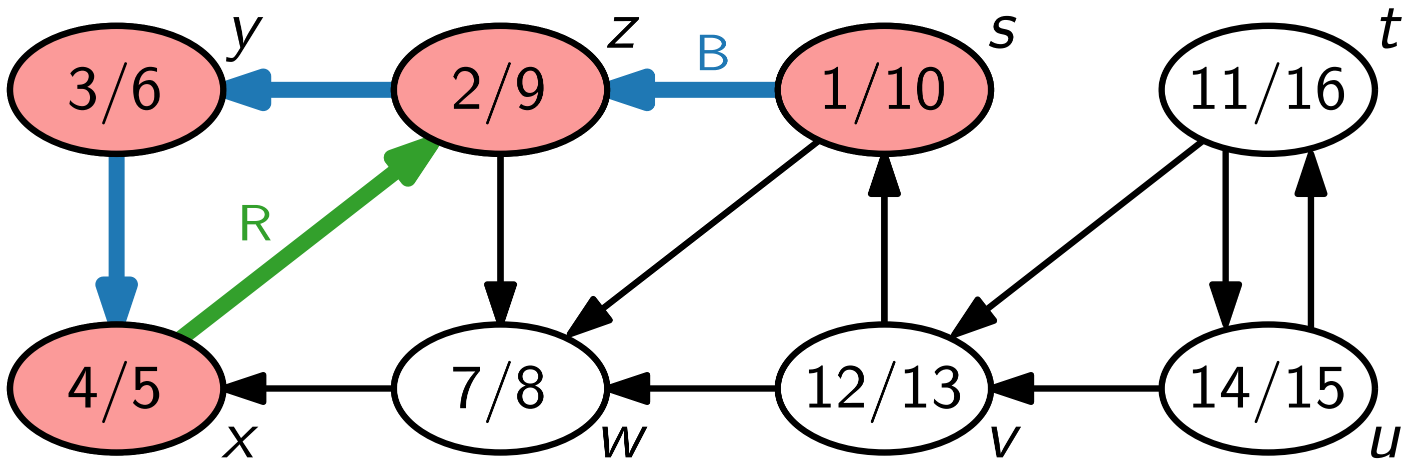


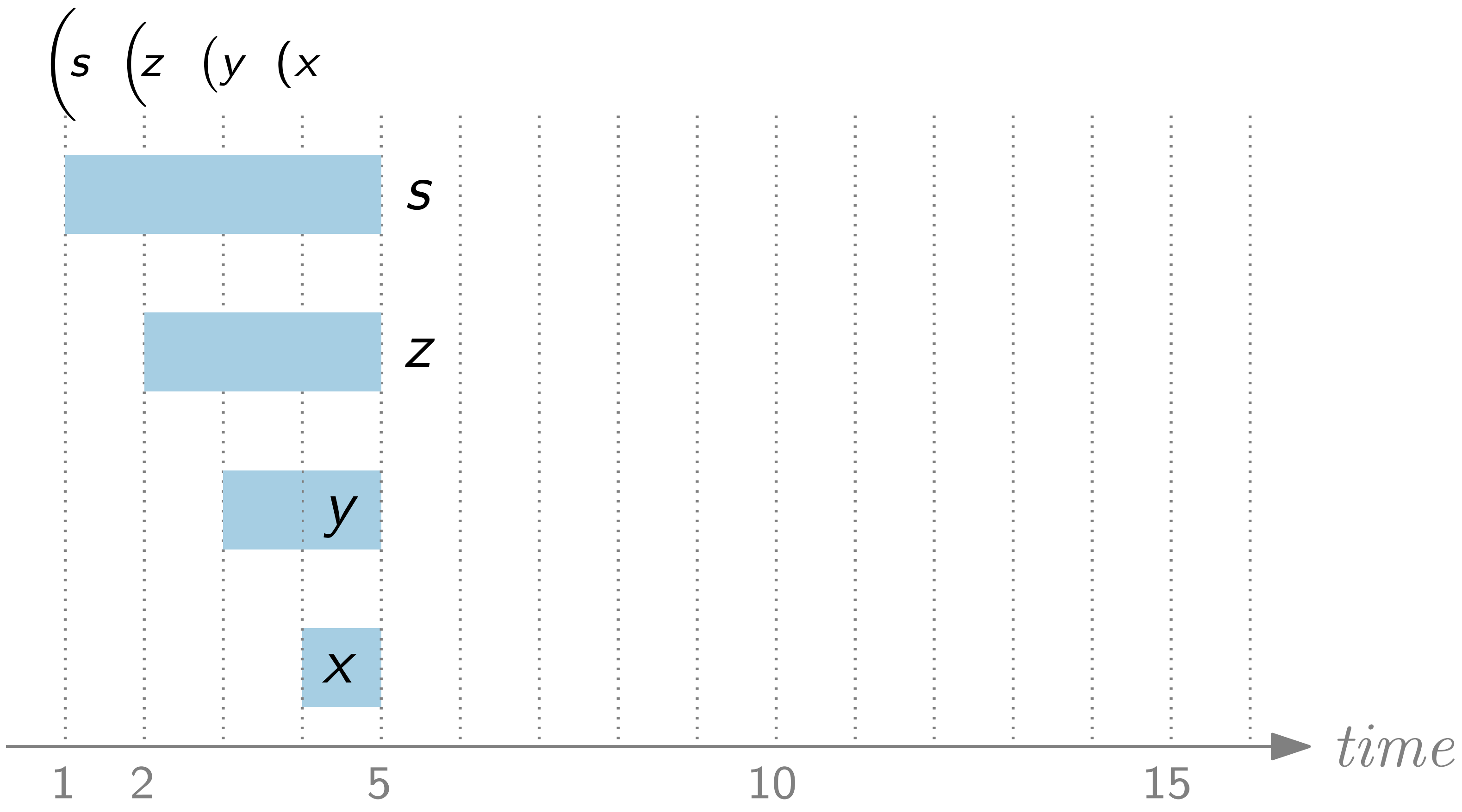
Tiefensuche – Eigenschaften



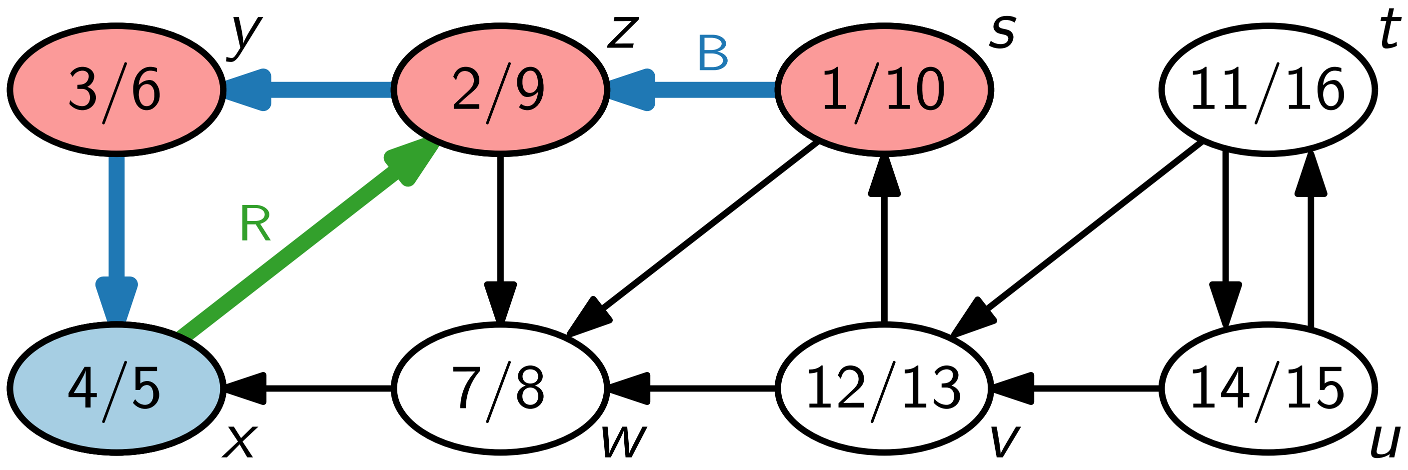


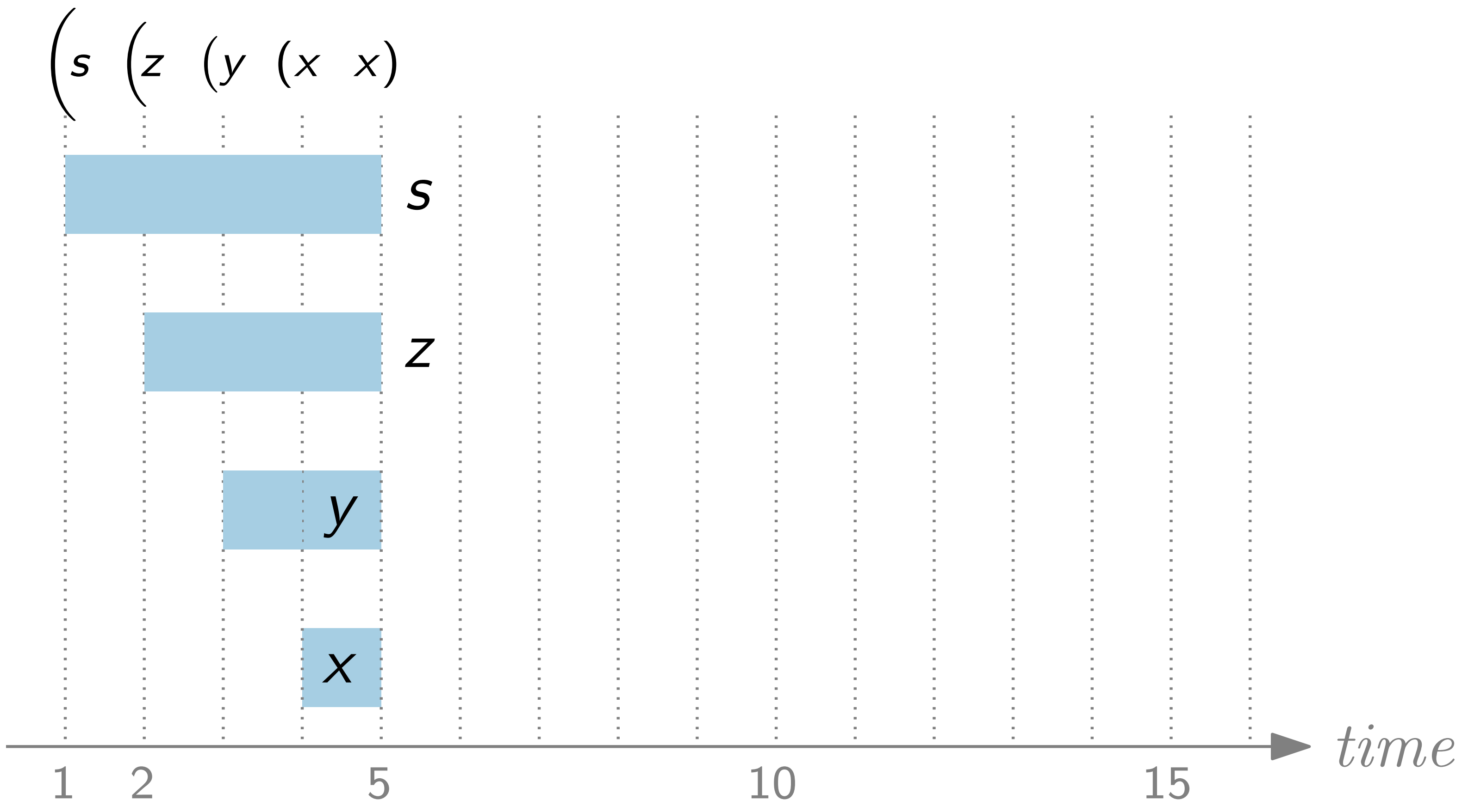
Tiefensuche – Eigenschaften



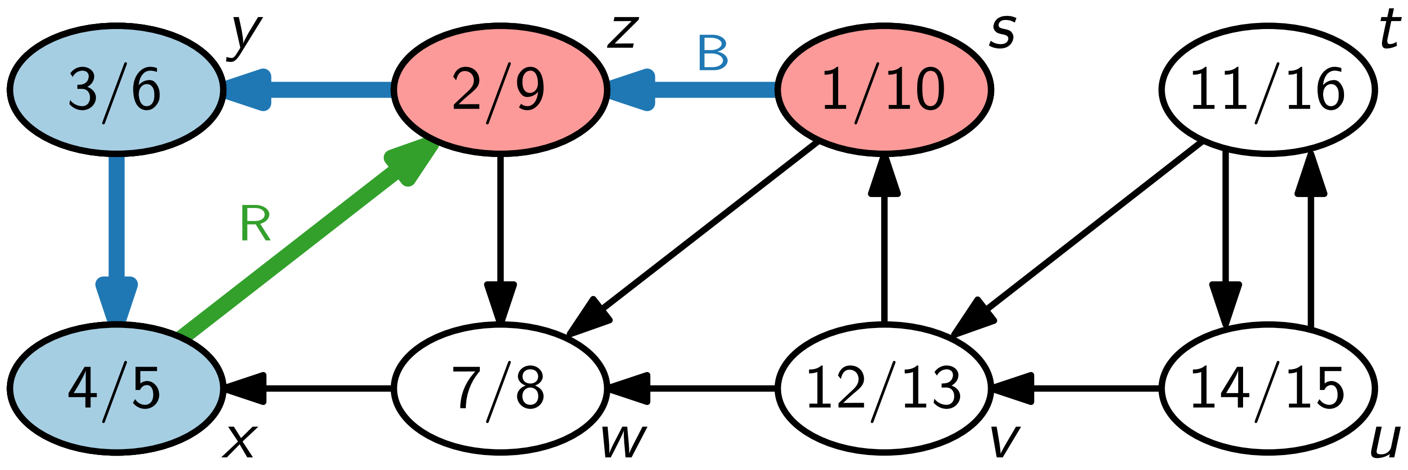


Tiefensuche – Eigenschaften

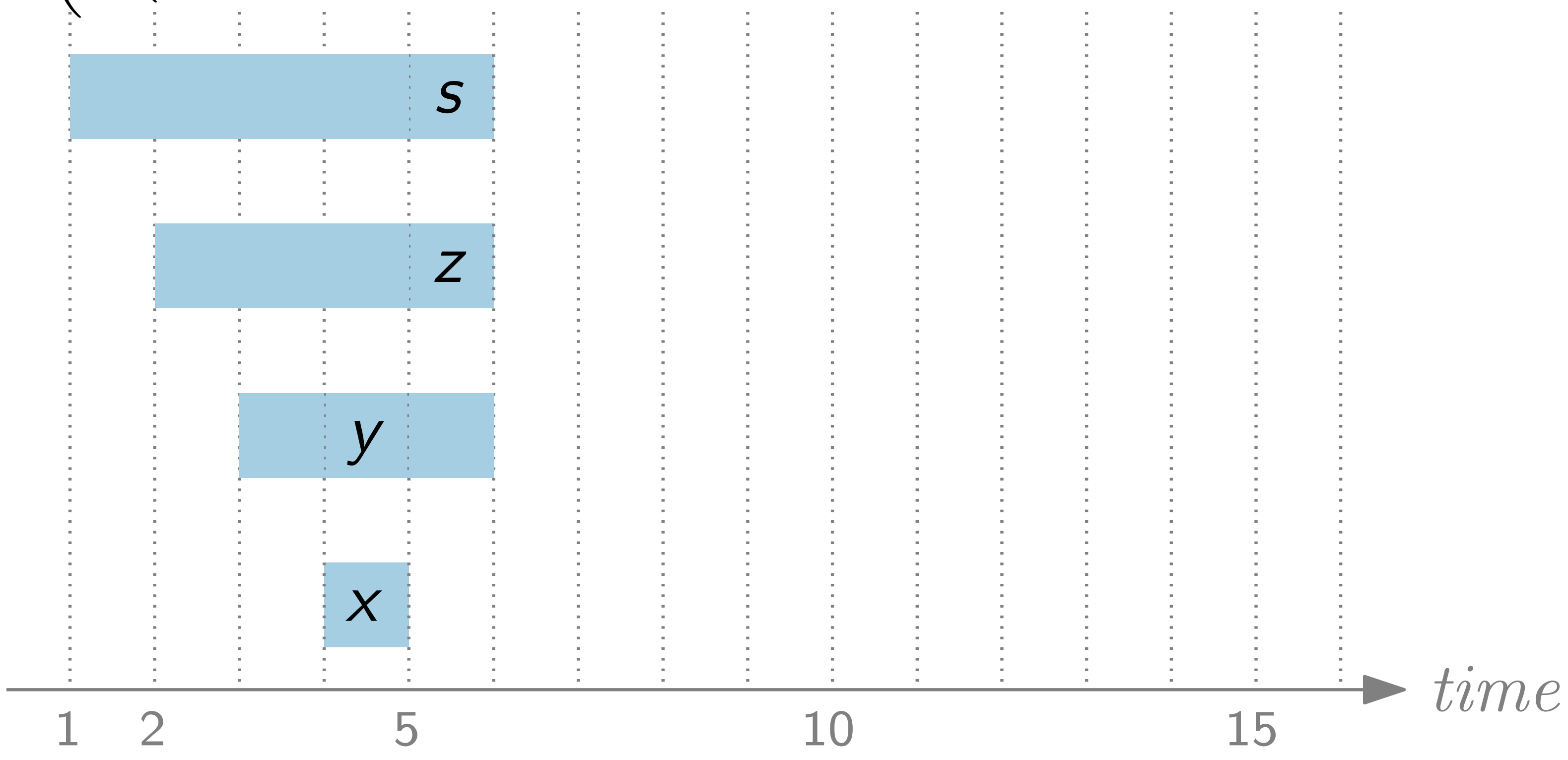




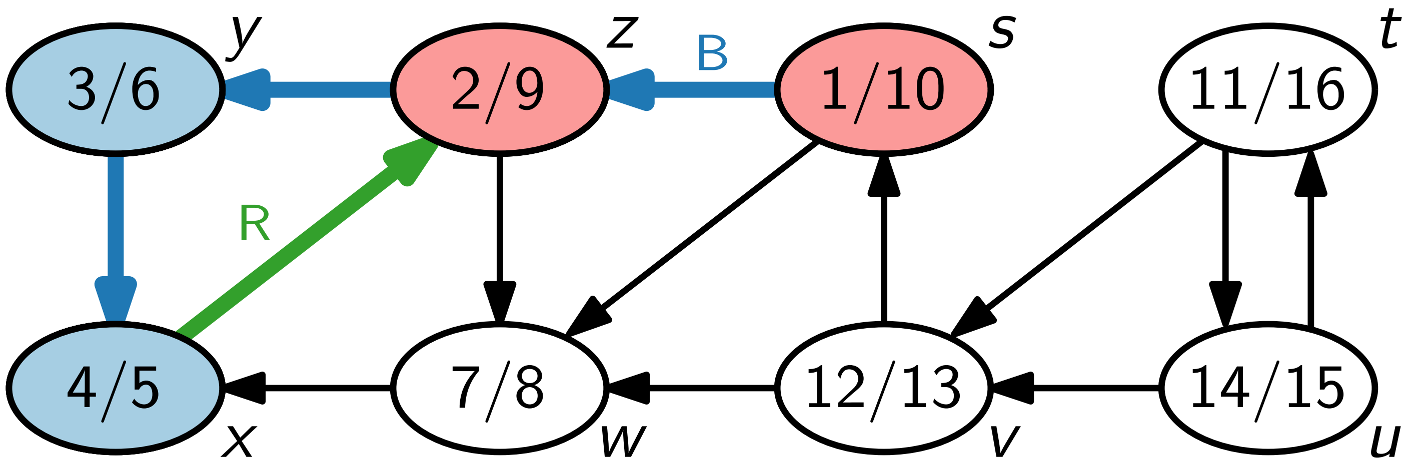
Tiefensuche – Eigenschaften

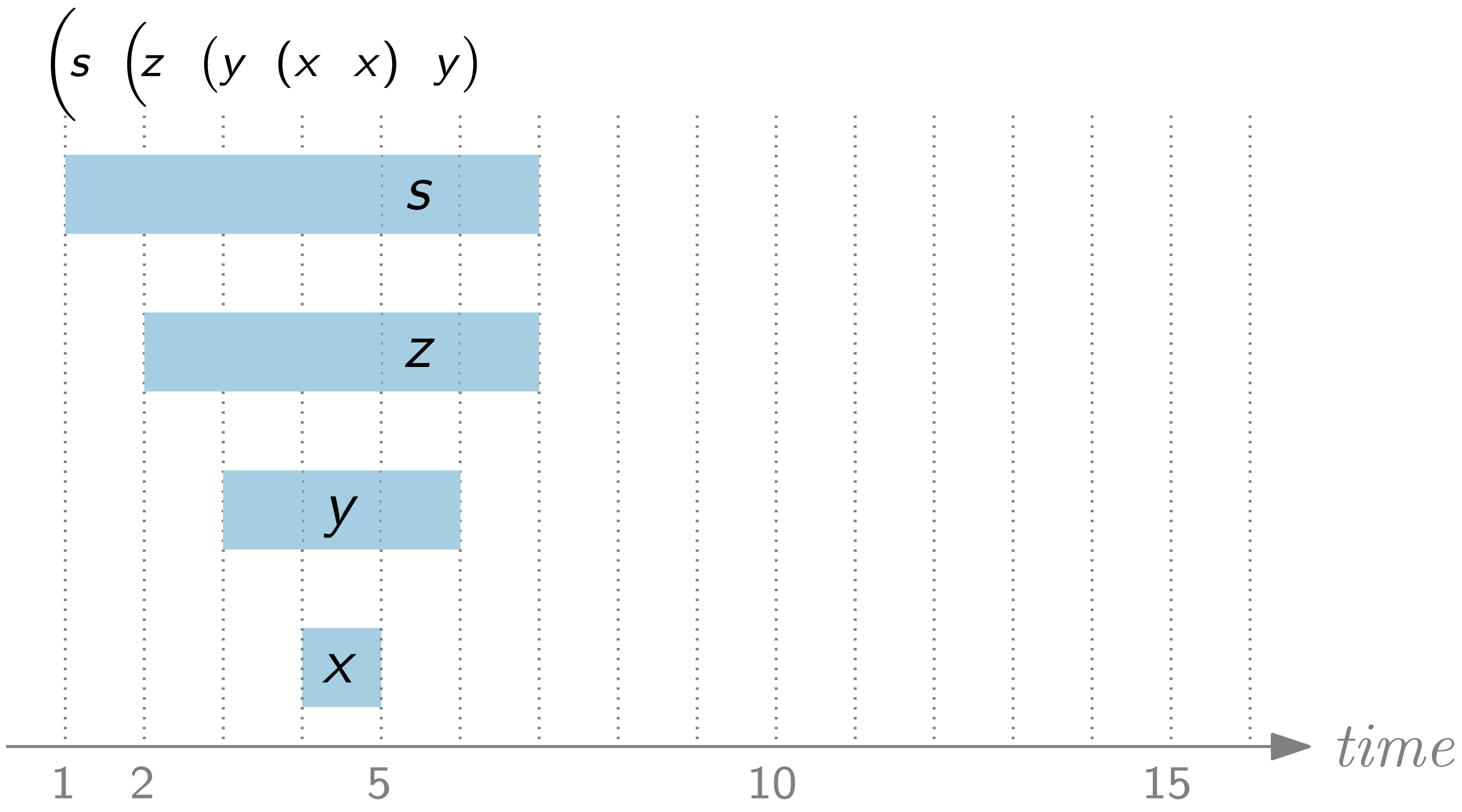


$\Big(s \ \Big(z \ \ (y \ \ (x \ \ x) \ \ y)$

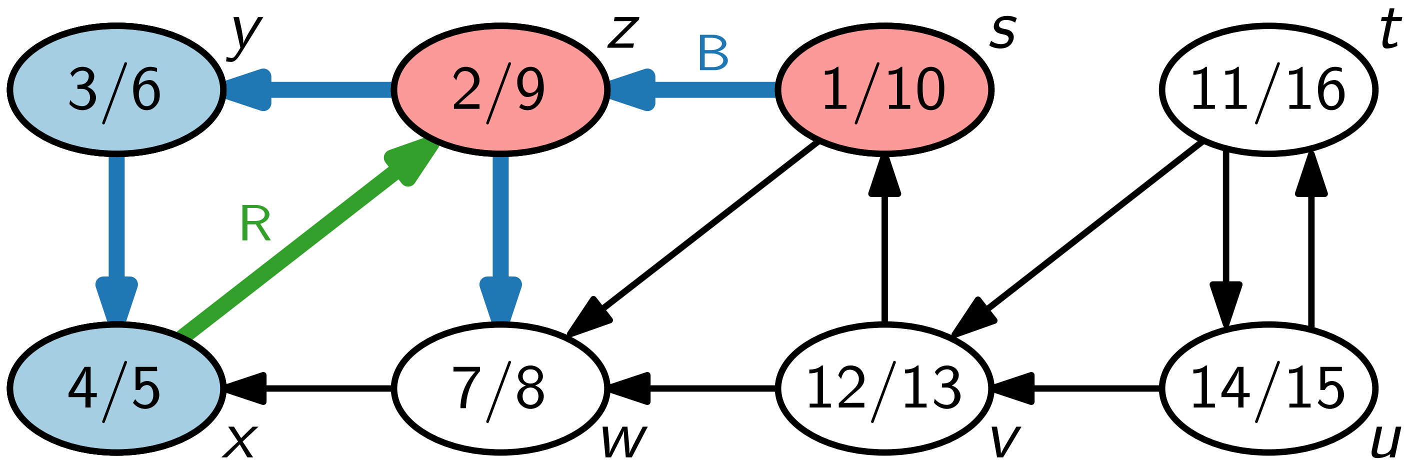


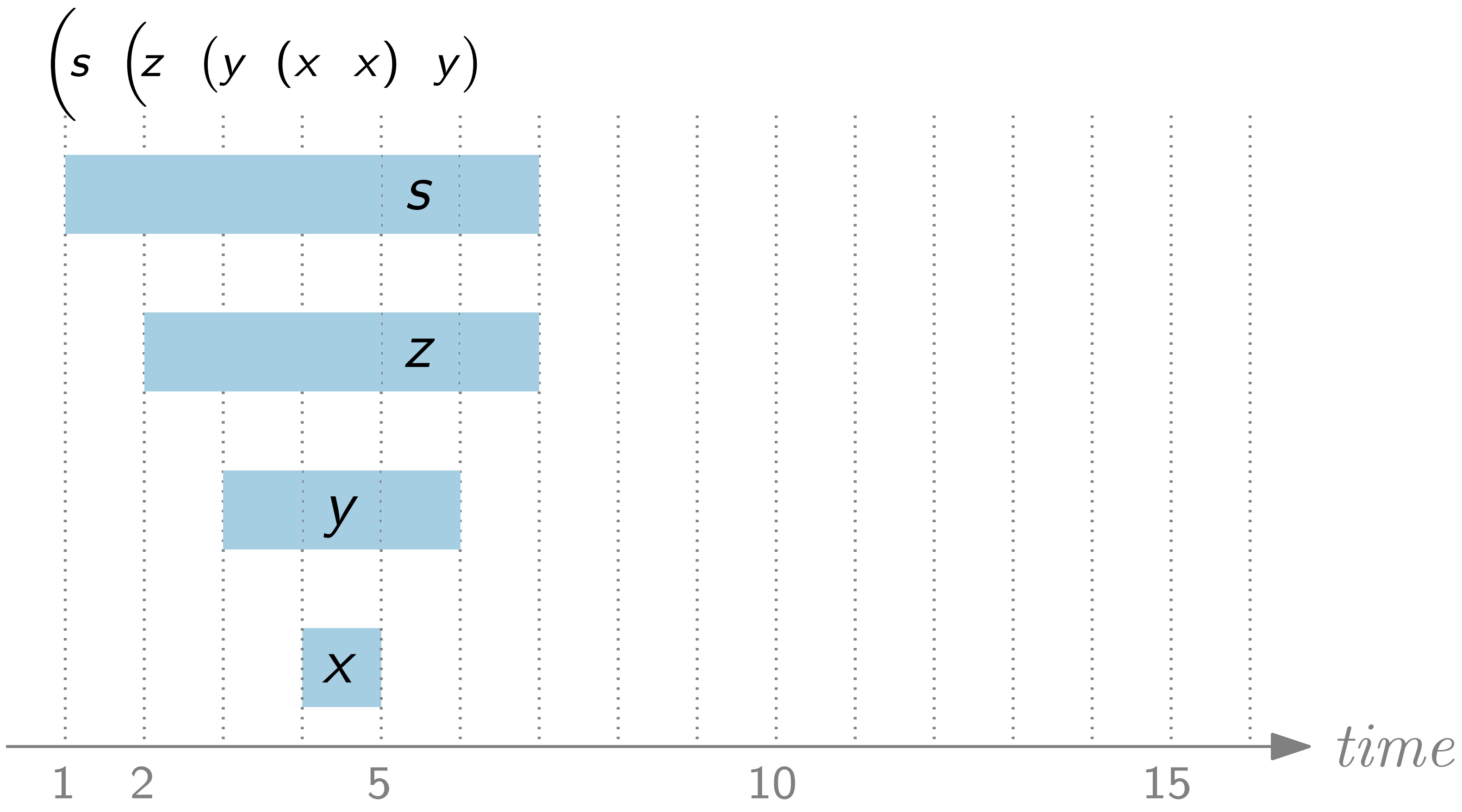
Tiefensuche – Eigenschaften



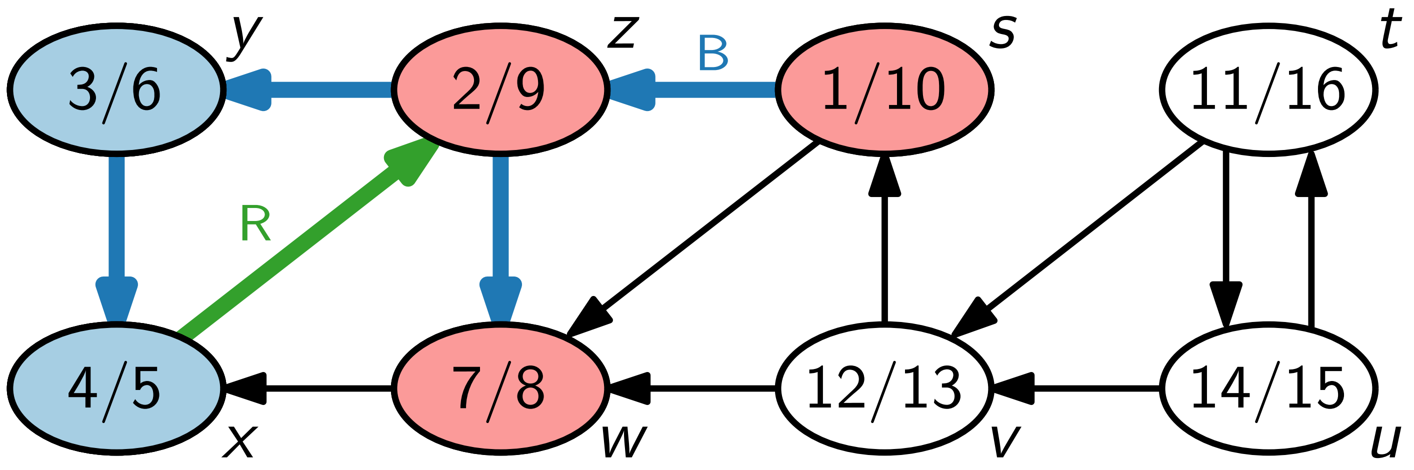


Tiefensuche – Eigenschaften

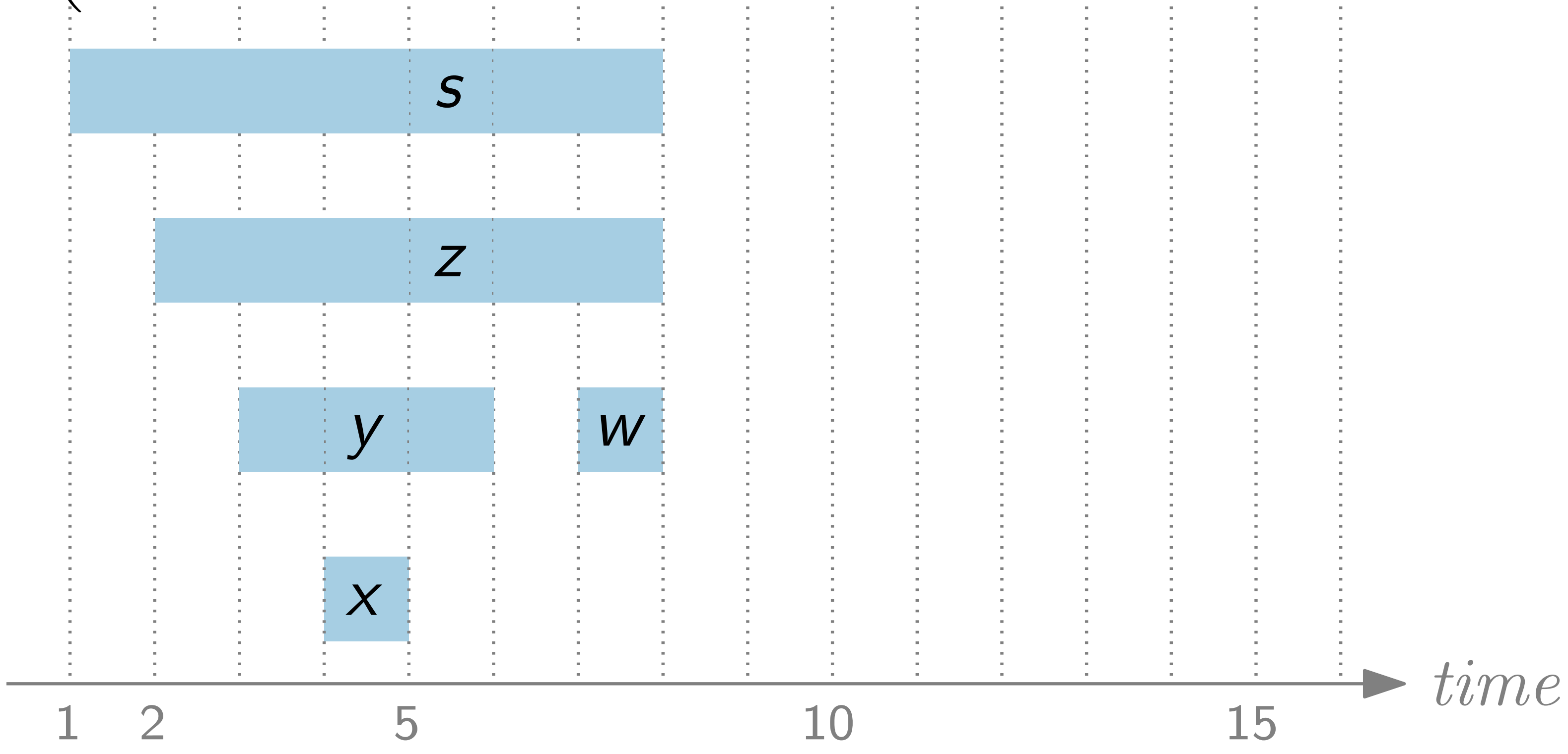




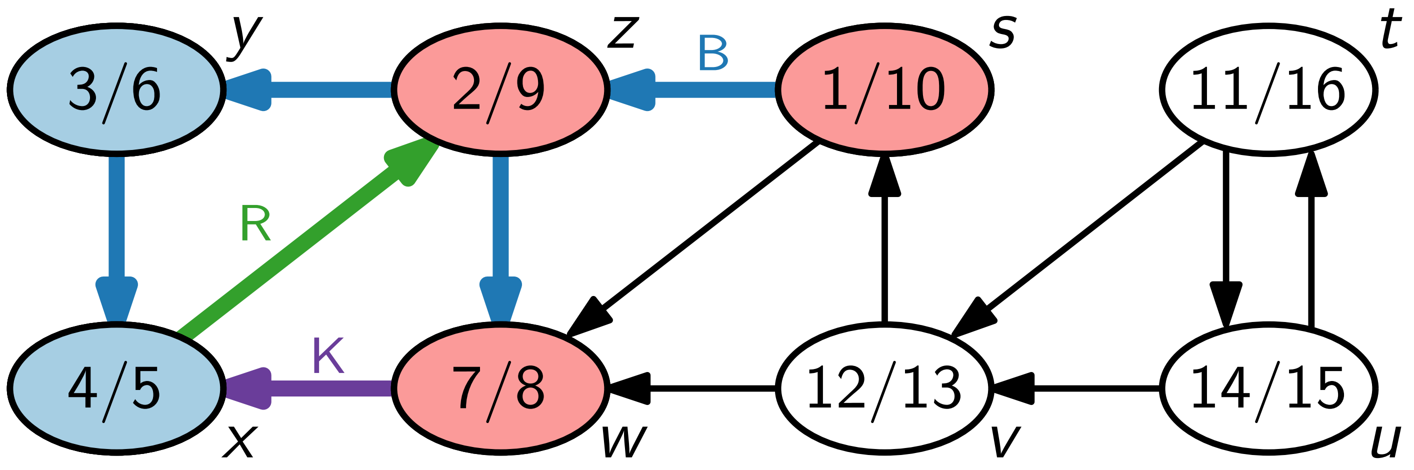
Tiefensuche – Eigenschaften

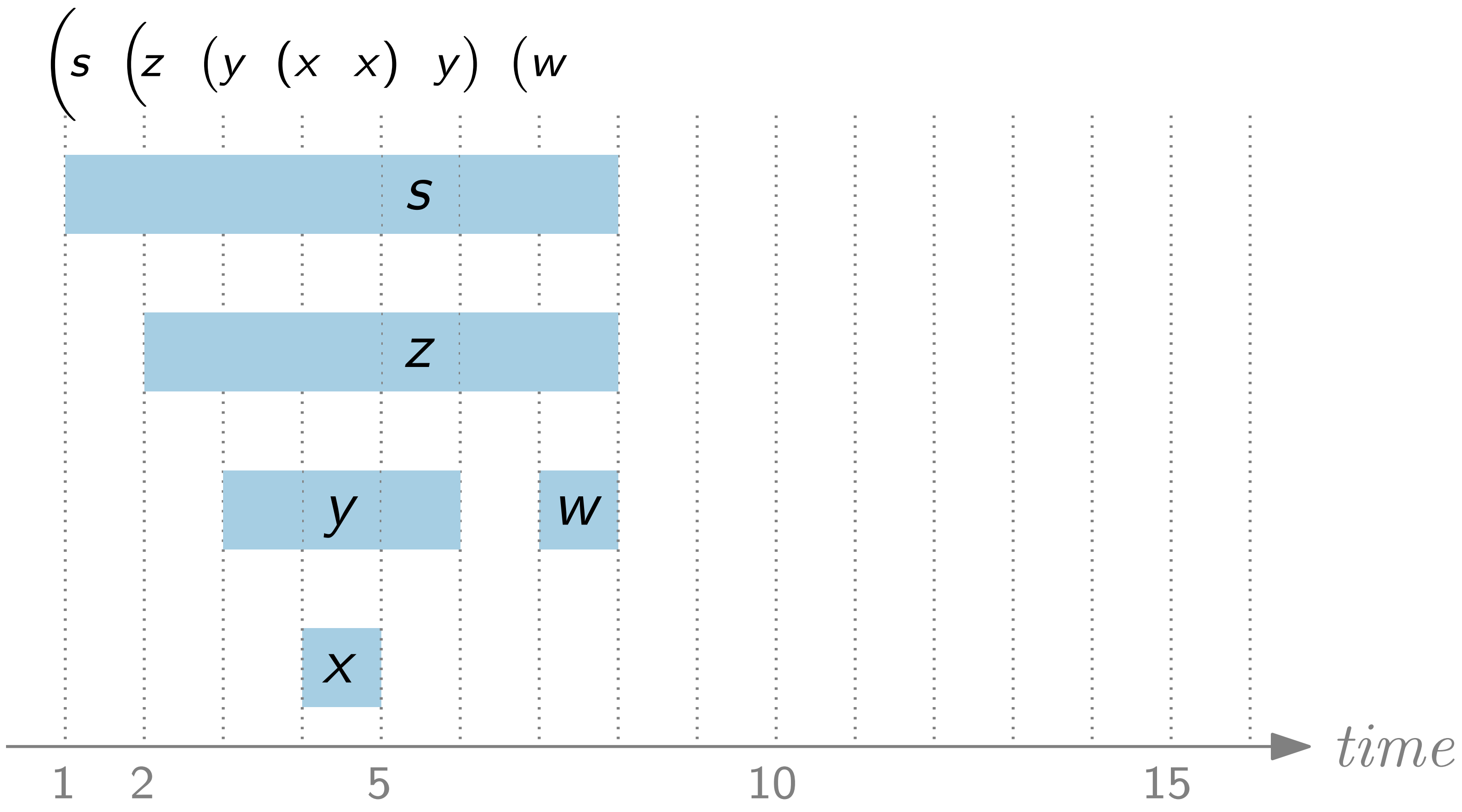


$\Big(s \;\Big(z \;\; (y \;\; (x \;\; x) \;\; y) \;\; (w$

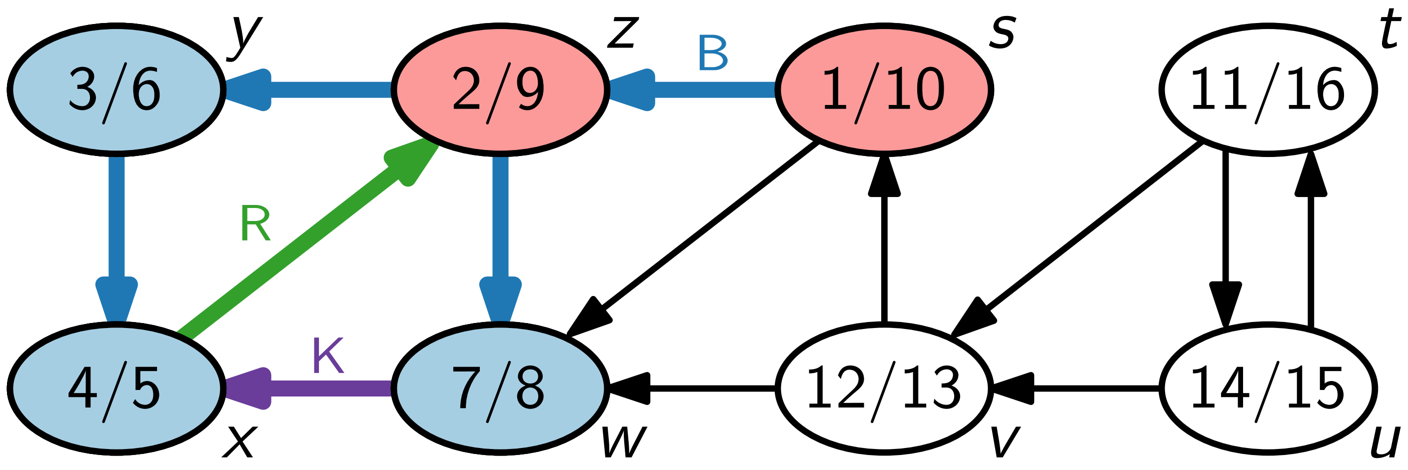


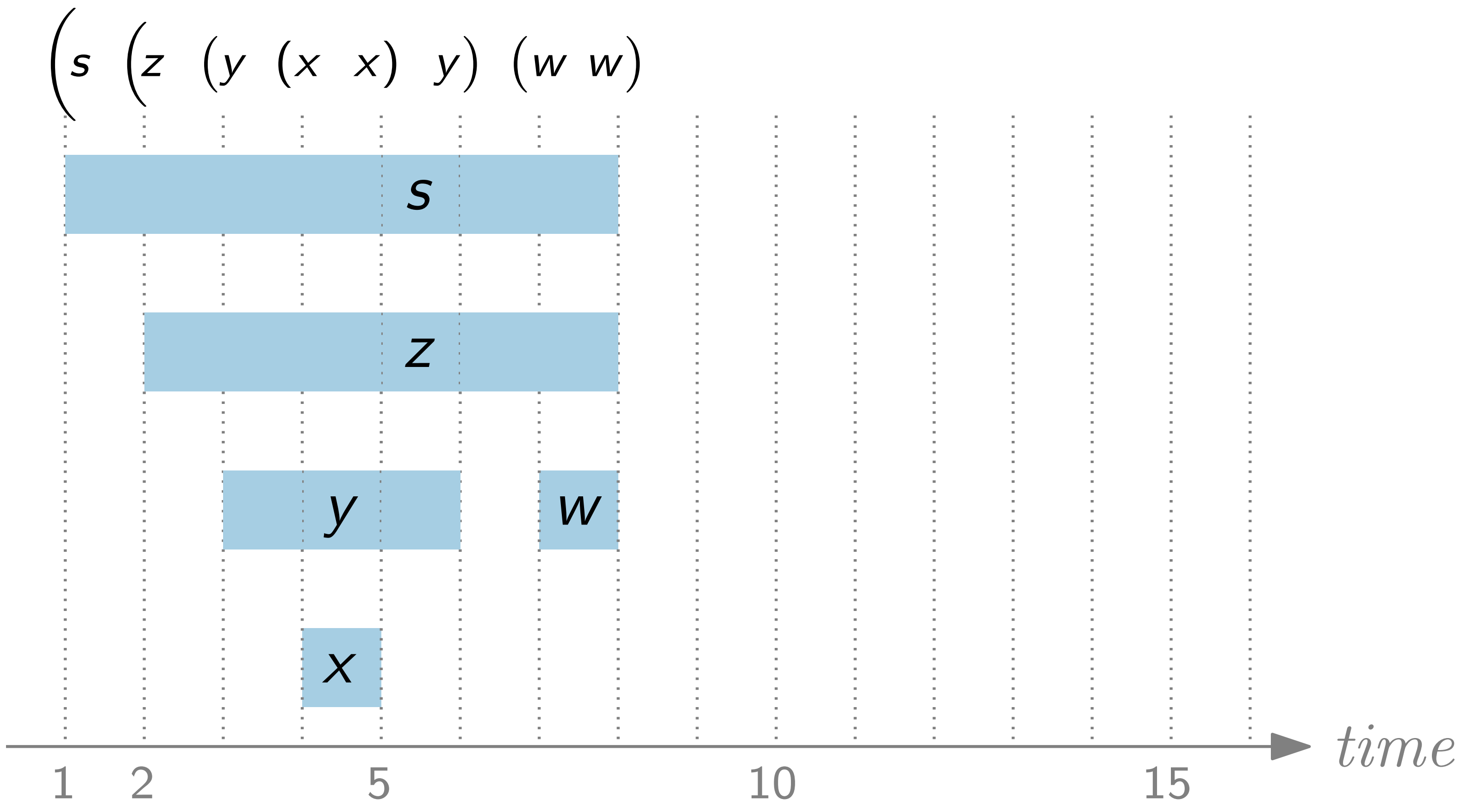
Tiefensuche – Eigenschaften



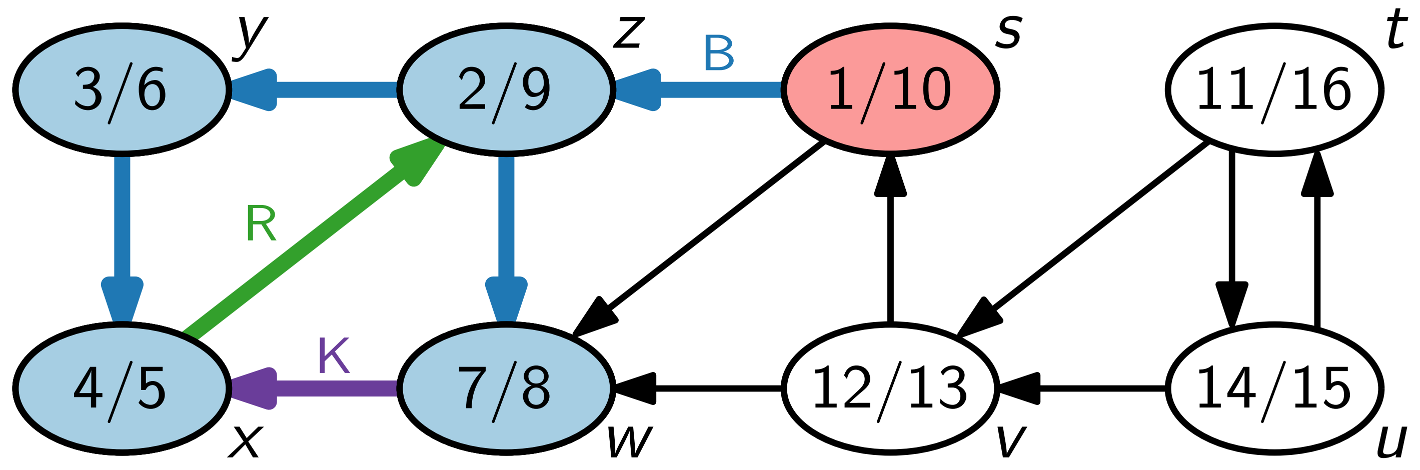


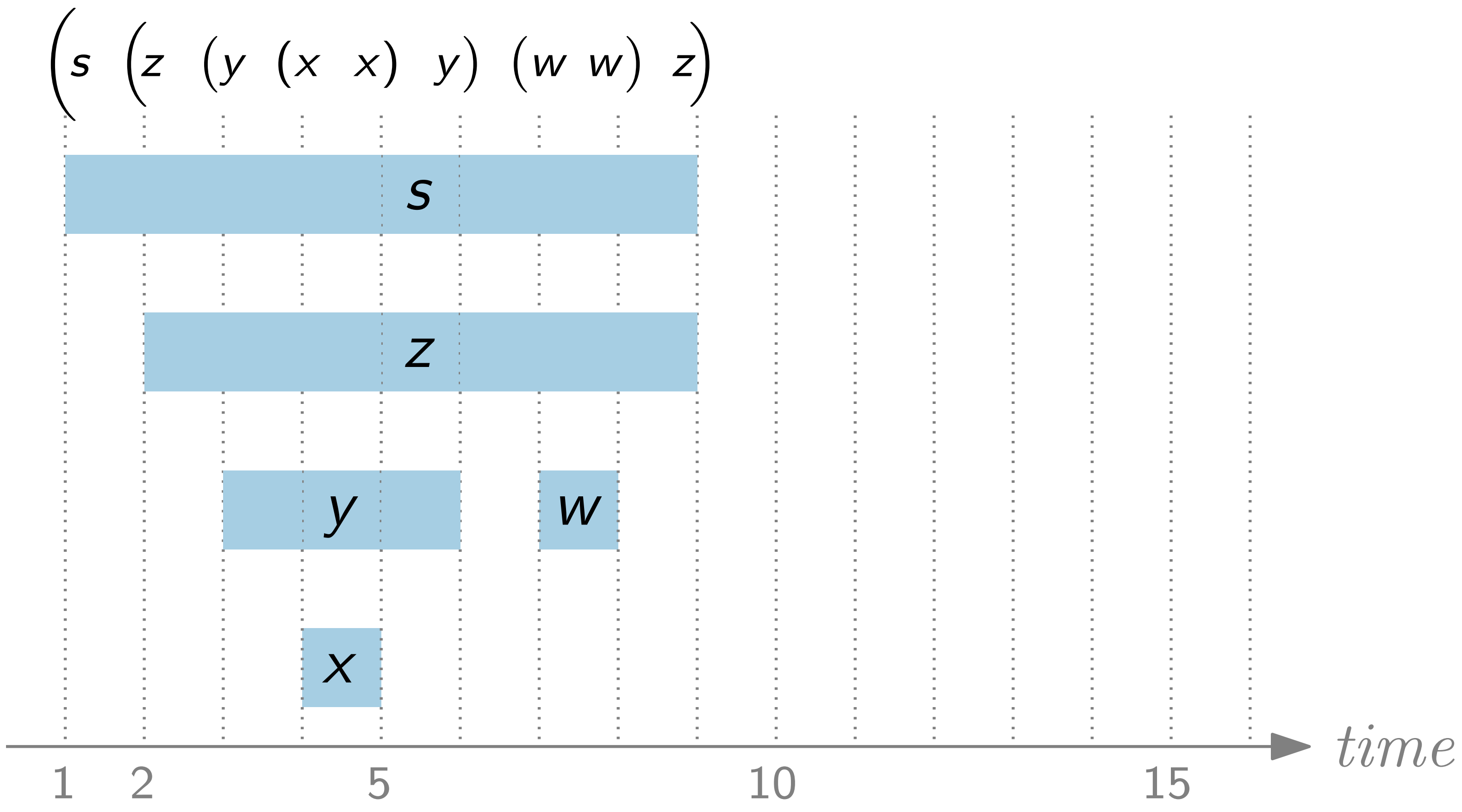
Tiefensuche – Eigenschaften



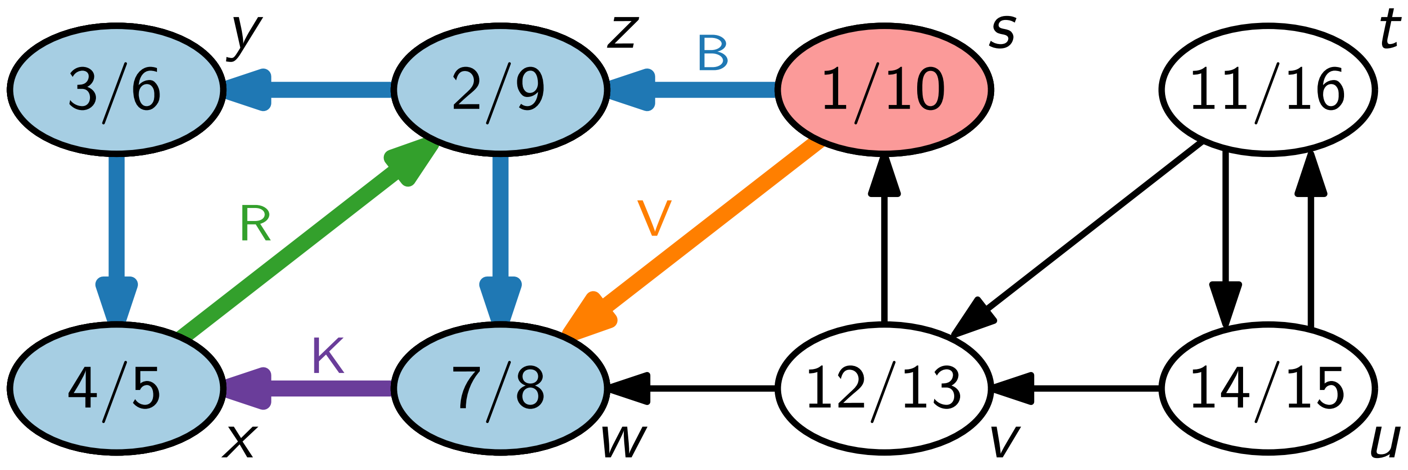


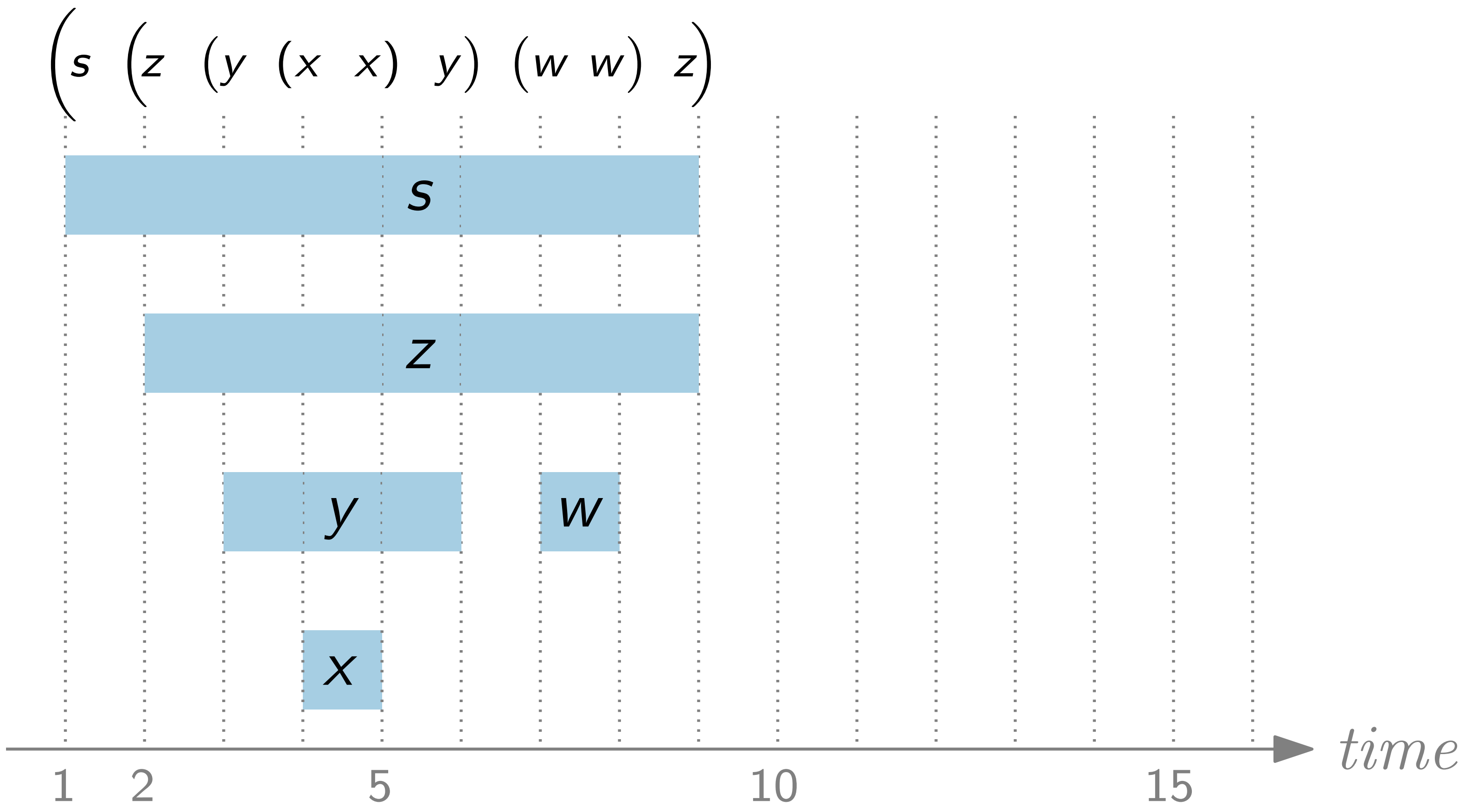
Tiefensuche – Eigenschaften



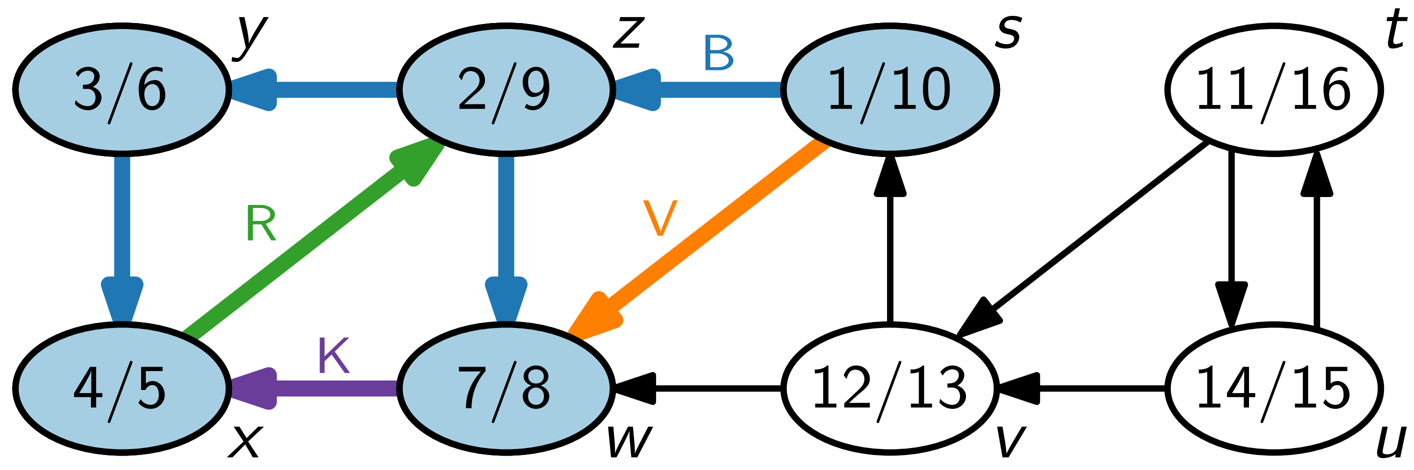


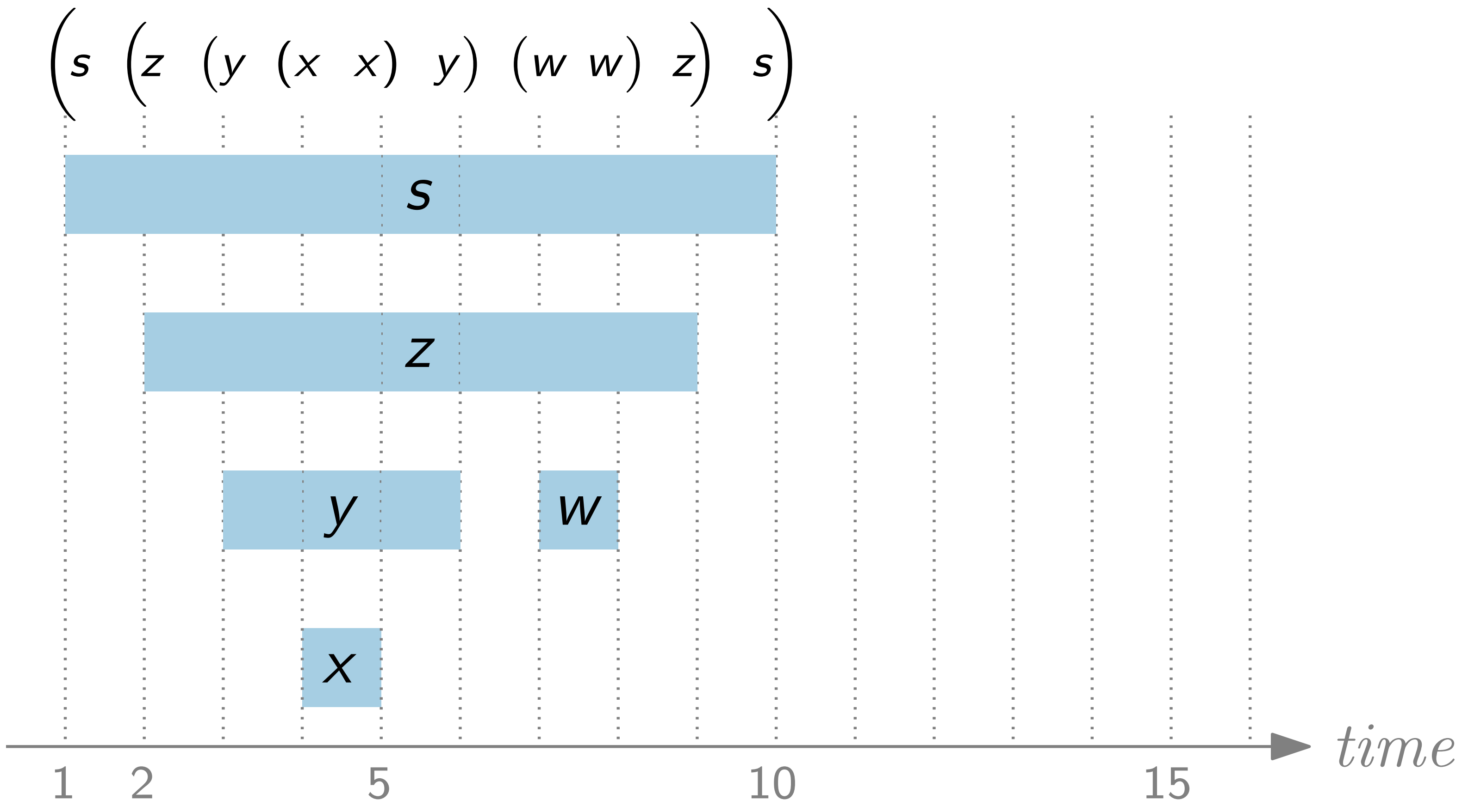
Tiefensuche – Eigenschaften



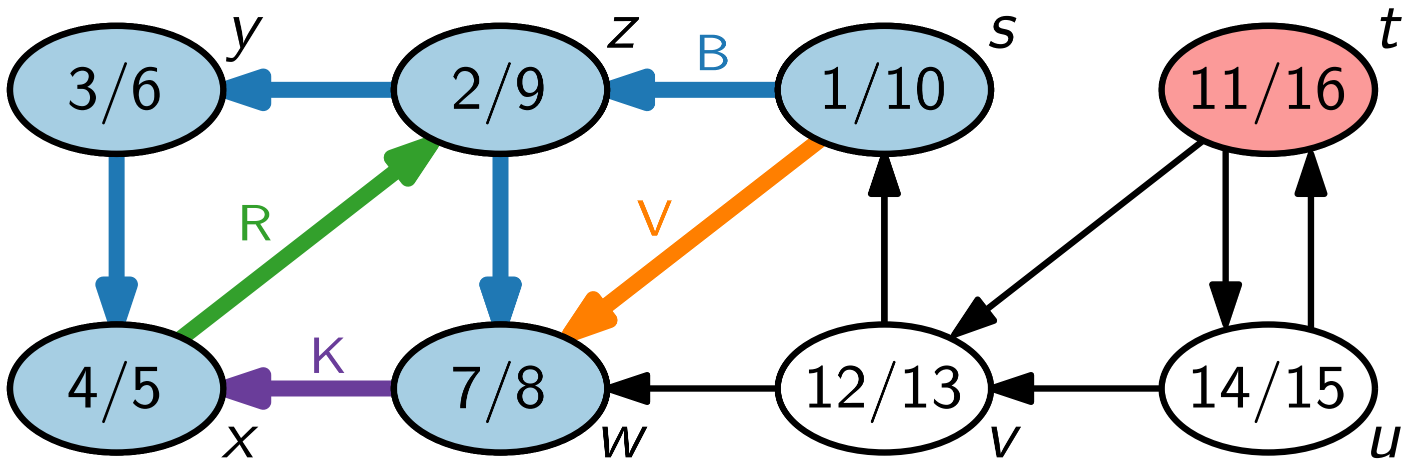


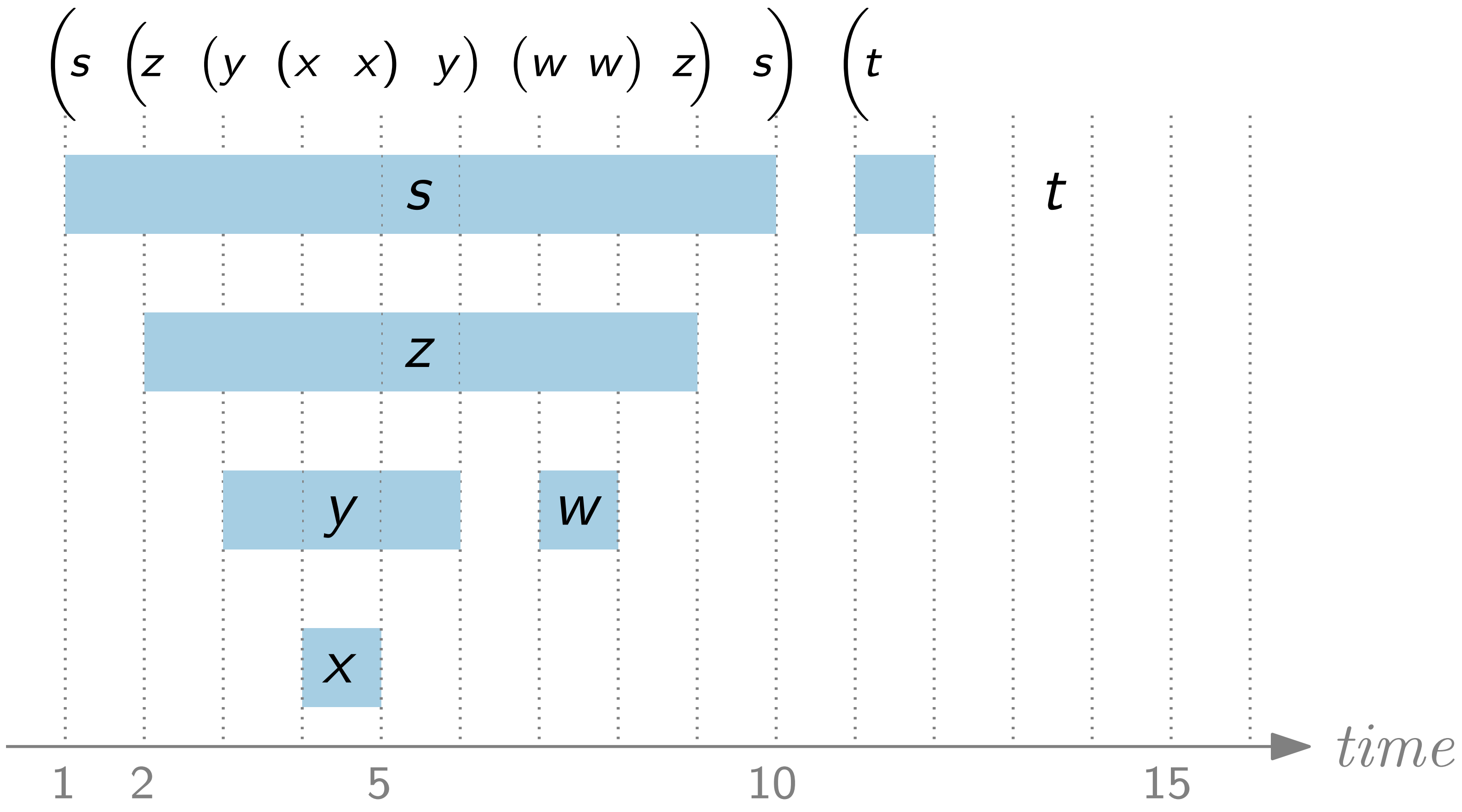
Tiefensuche – Eigenschaften



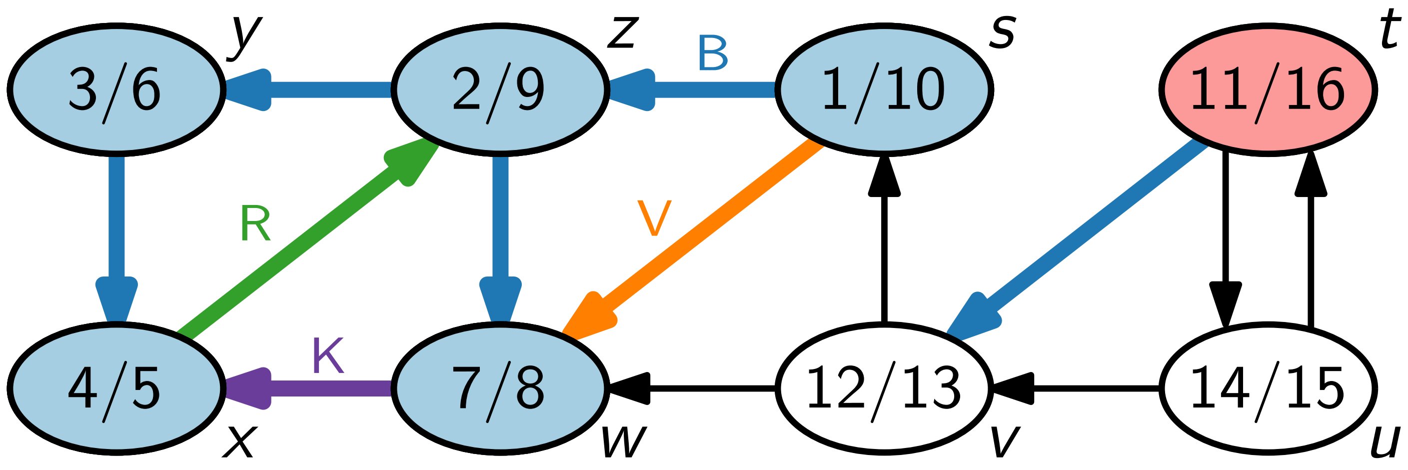


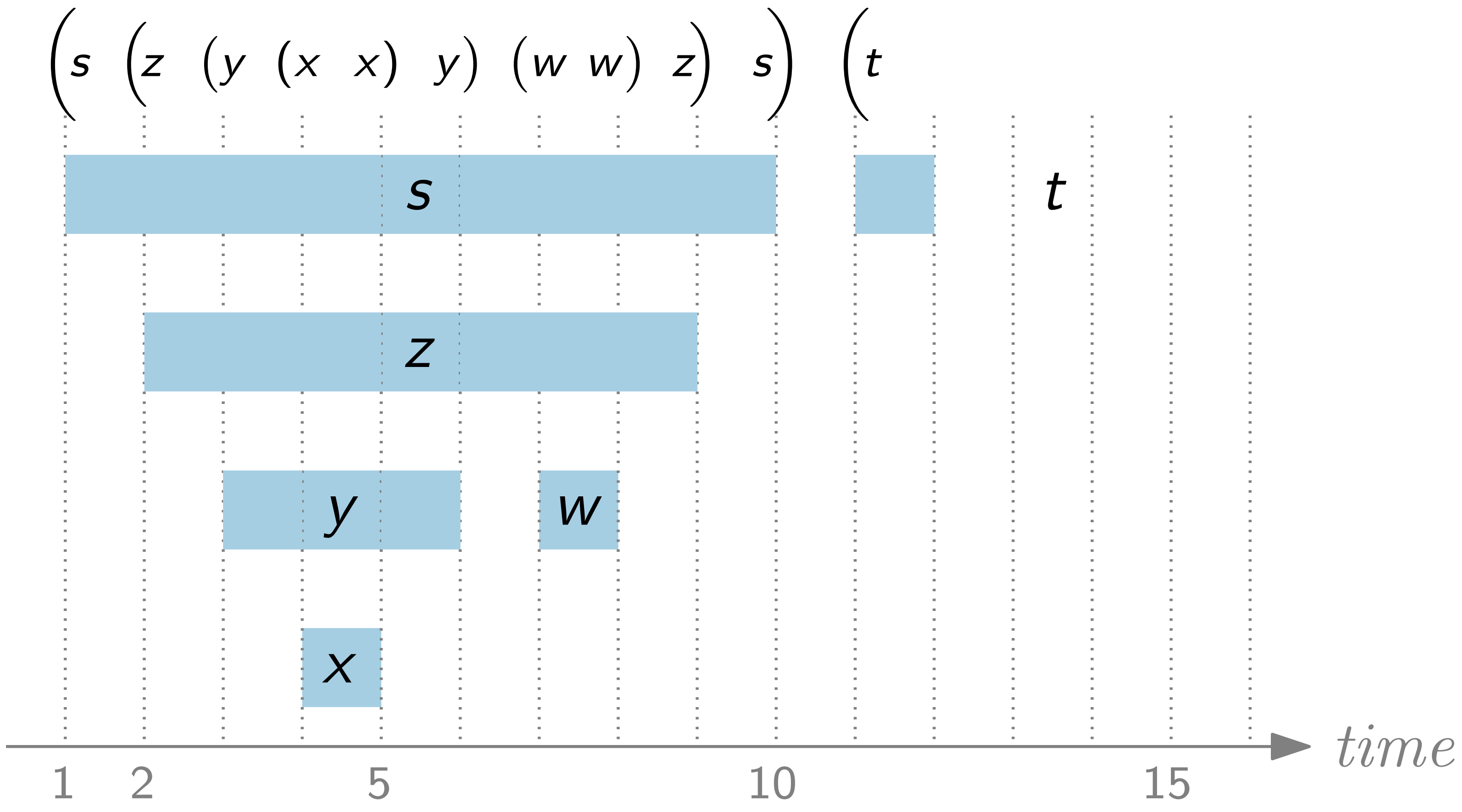
Tiefensuche – Eigenschaften



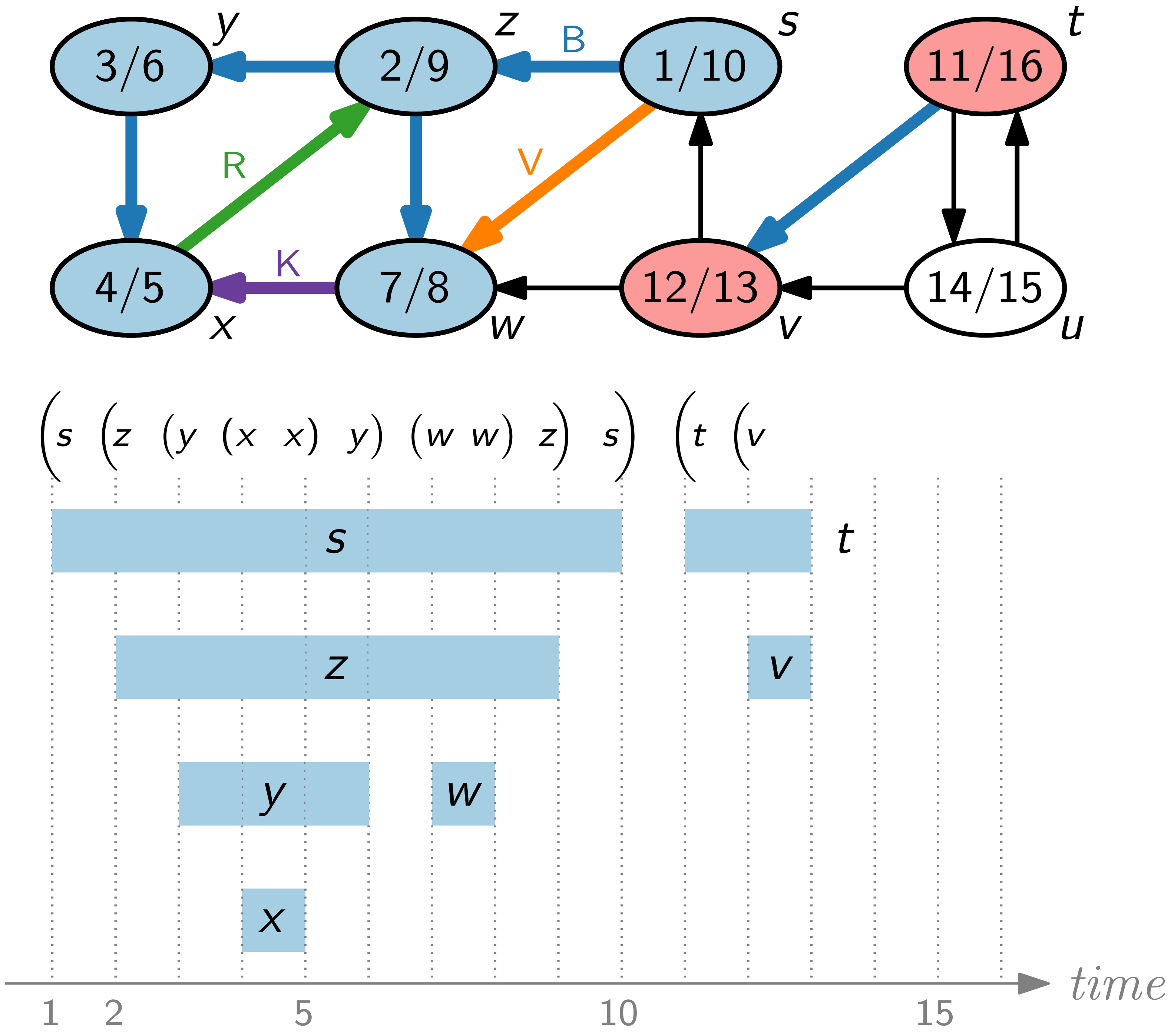


Tiefensuche – Eigenschaften

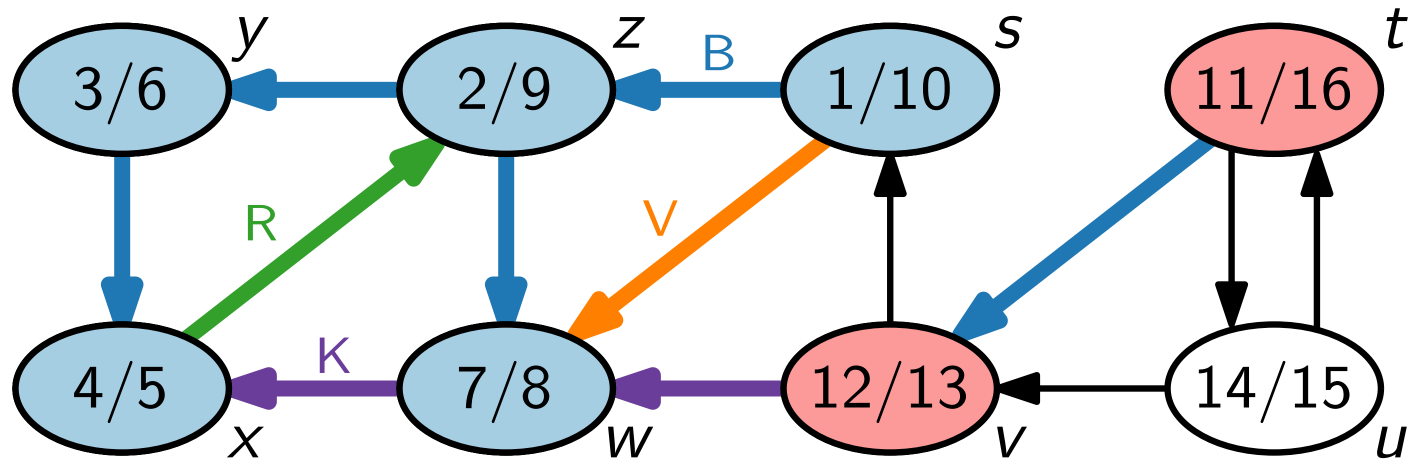


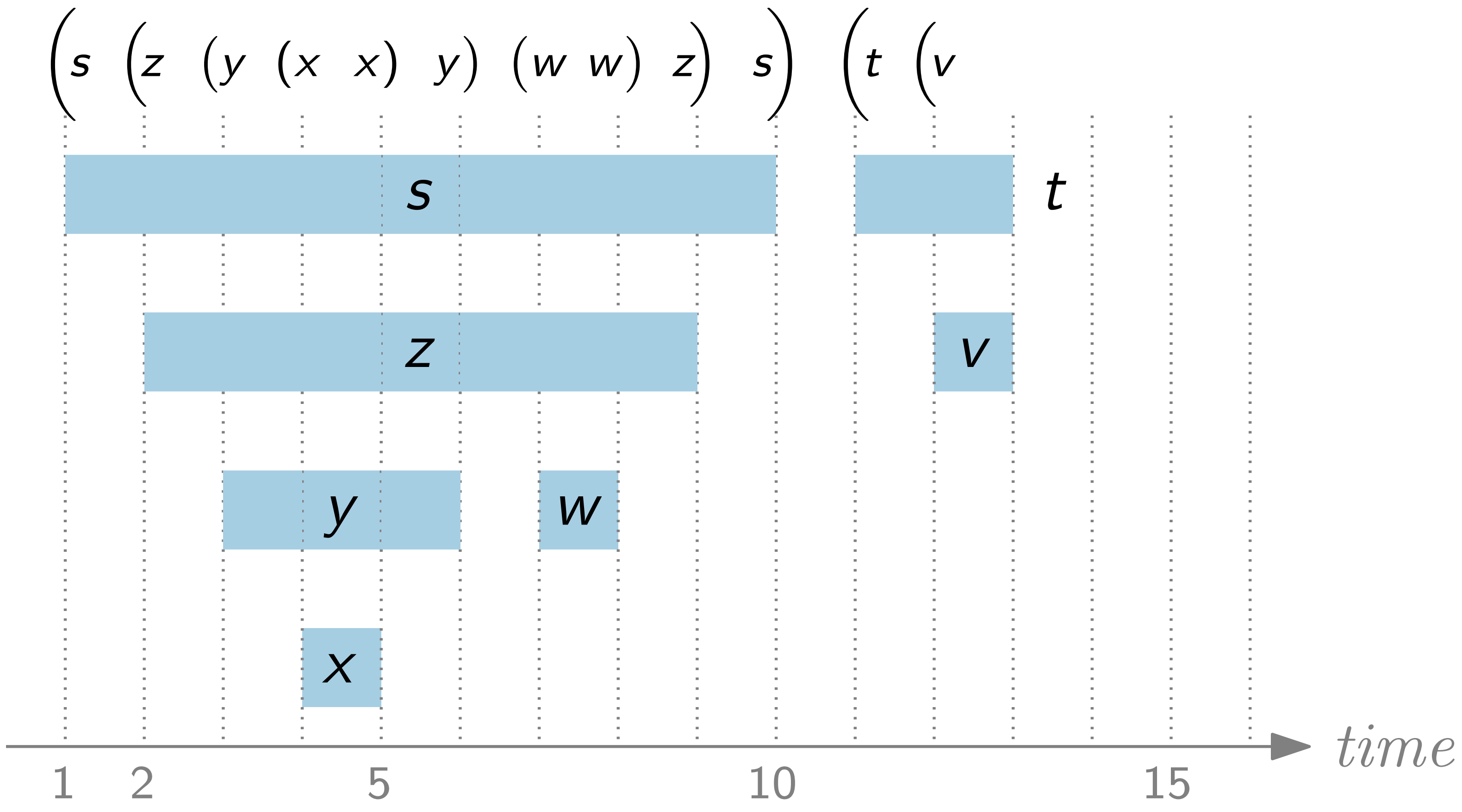


Tiefensuche – Eigenschaften

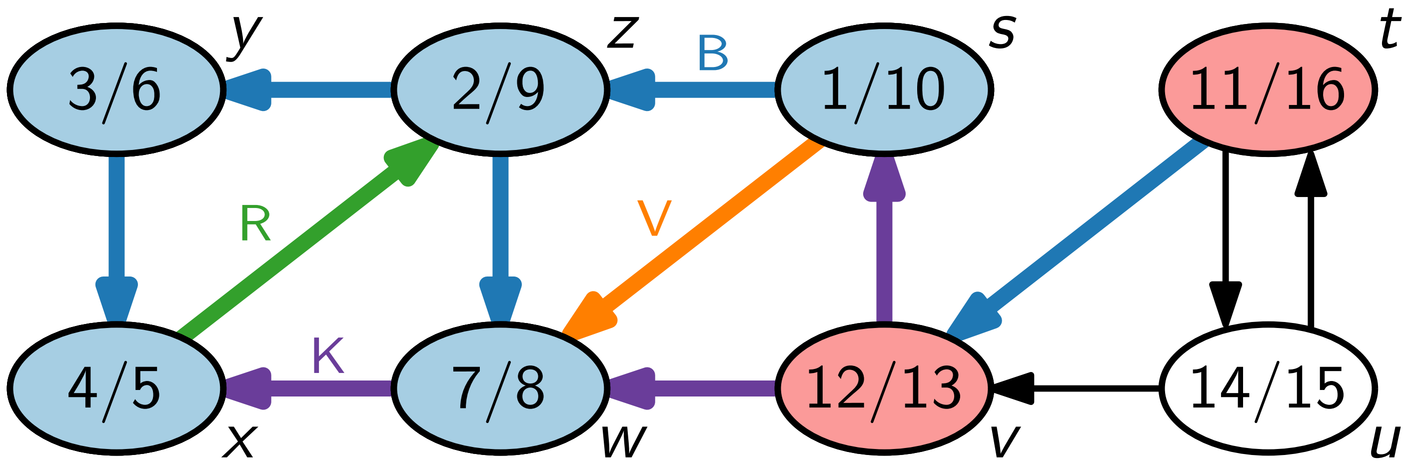


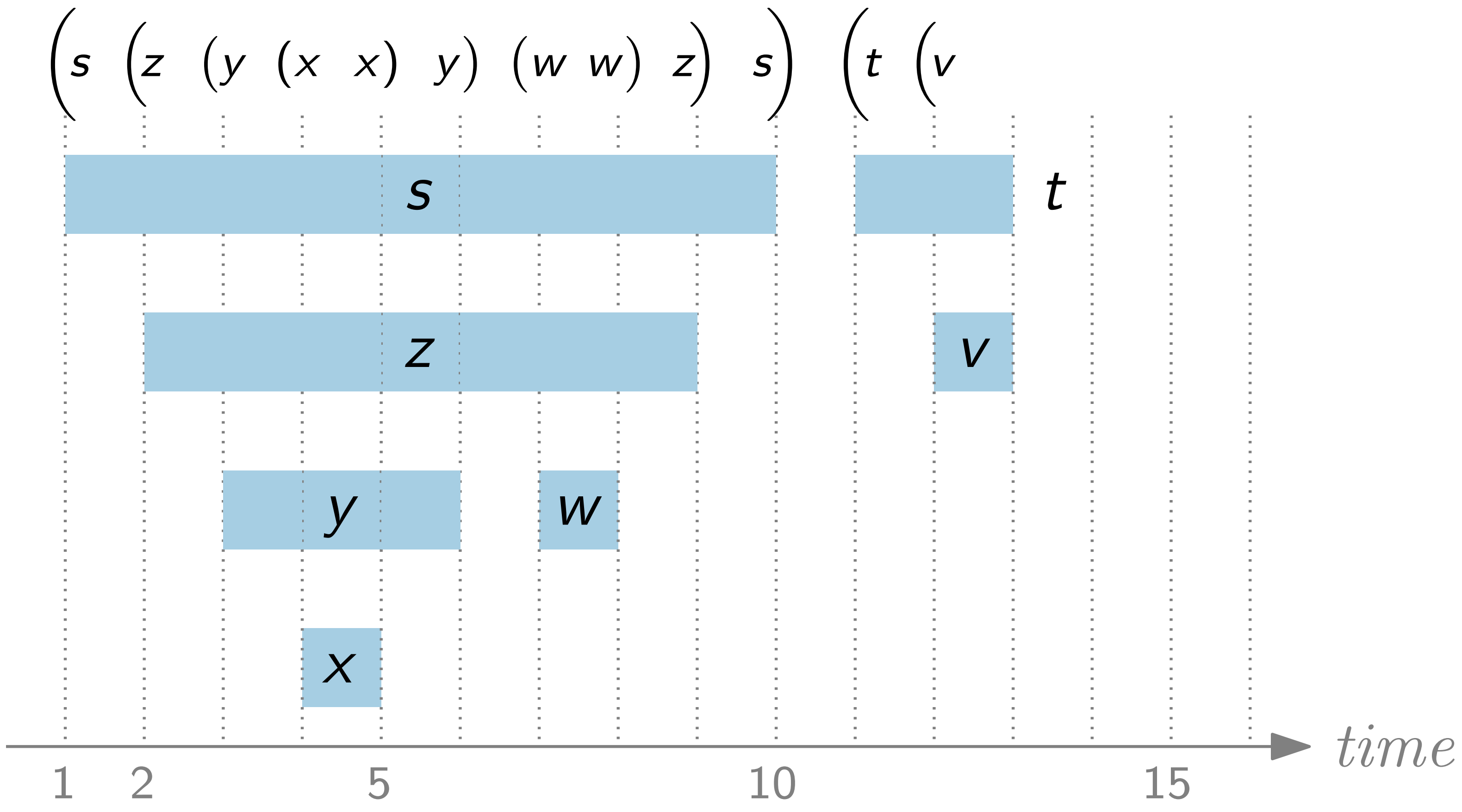
Tiefensuche – Eigenschaften



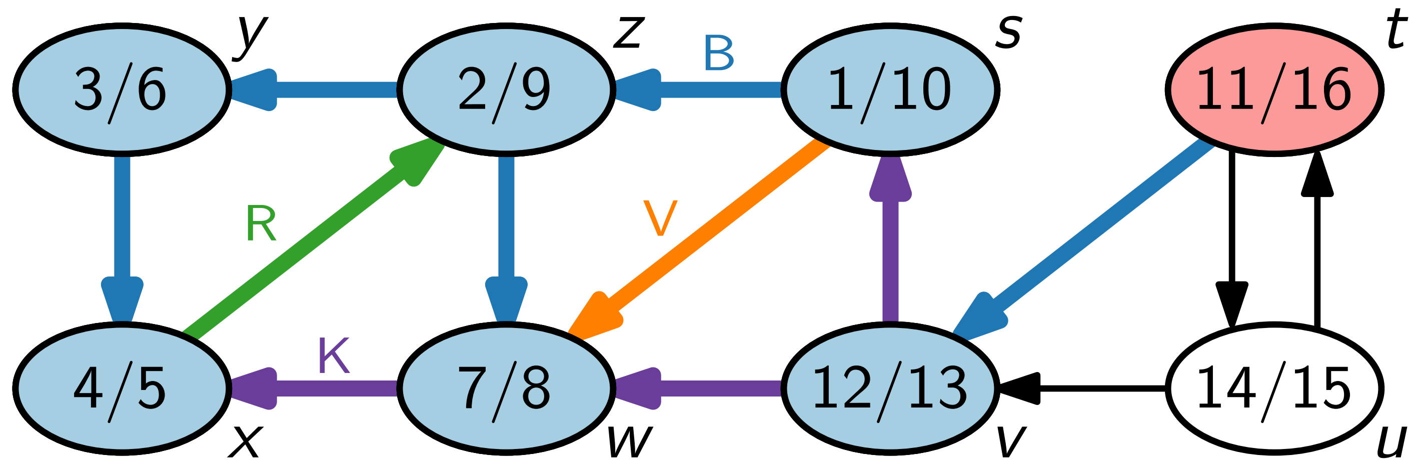


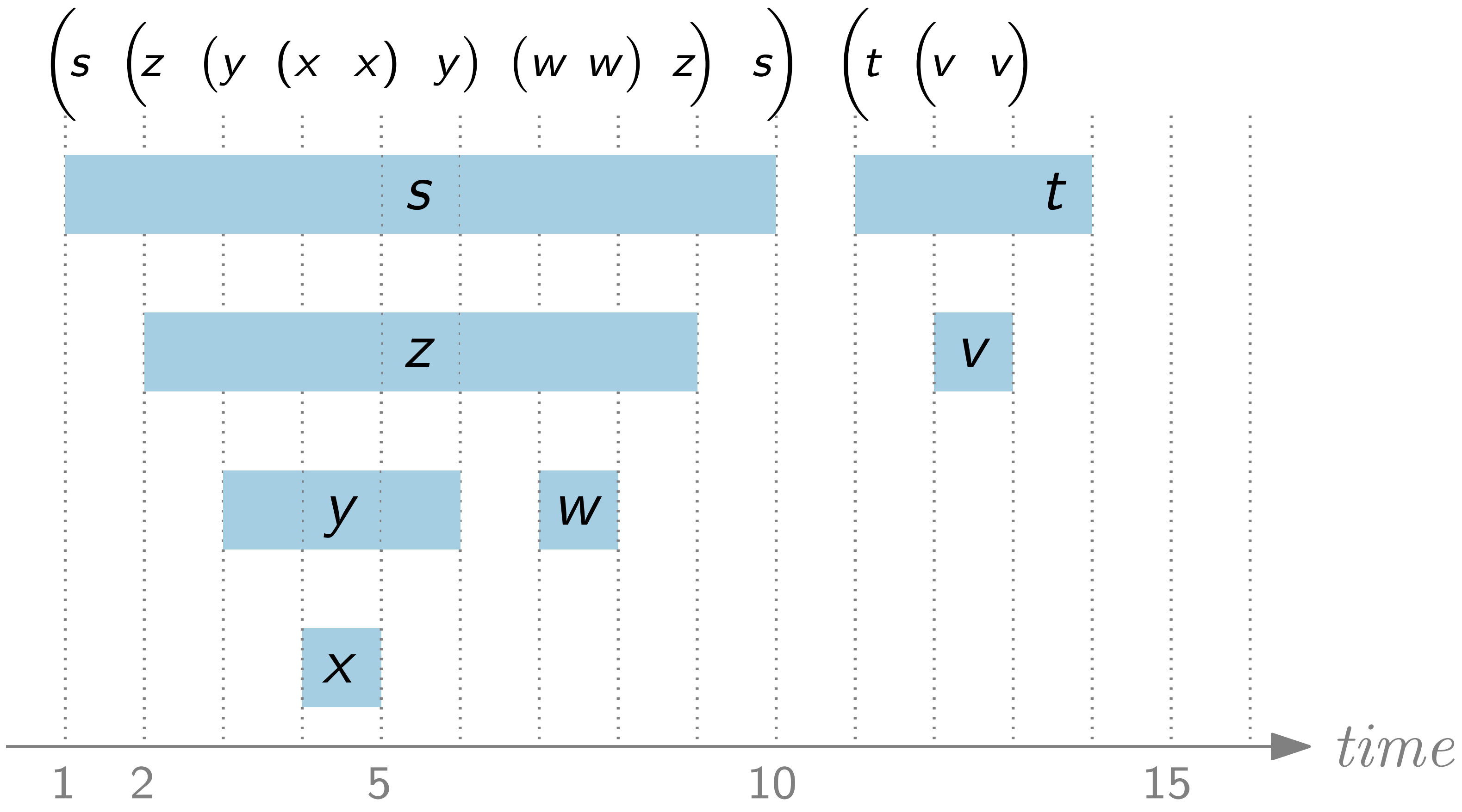
Tiefensuche – Eigenschaften



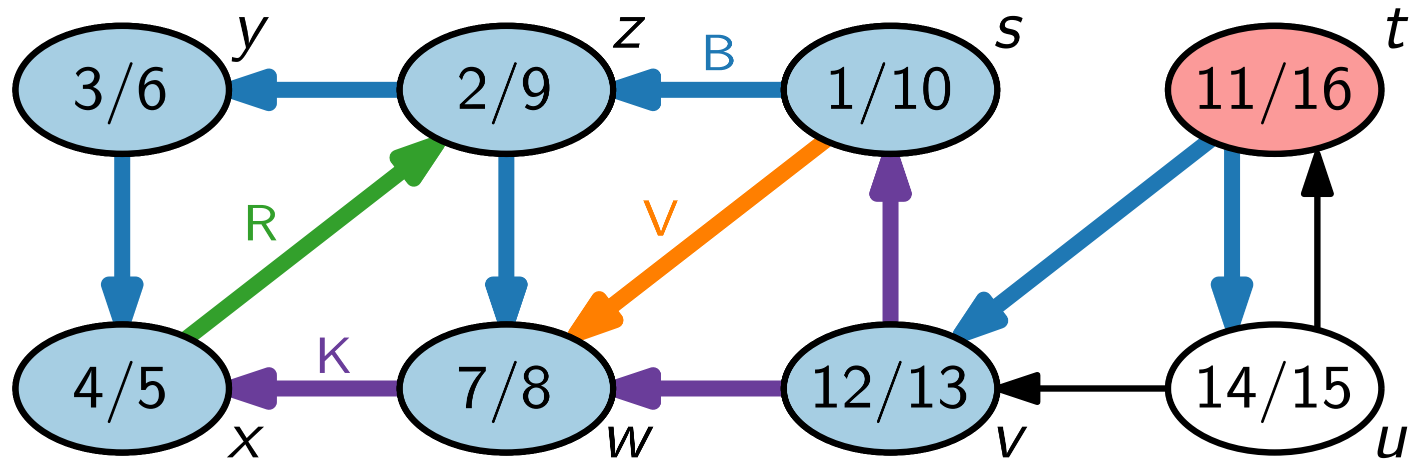


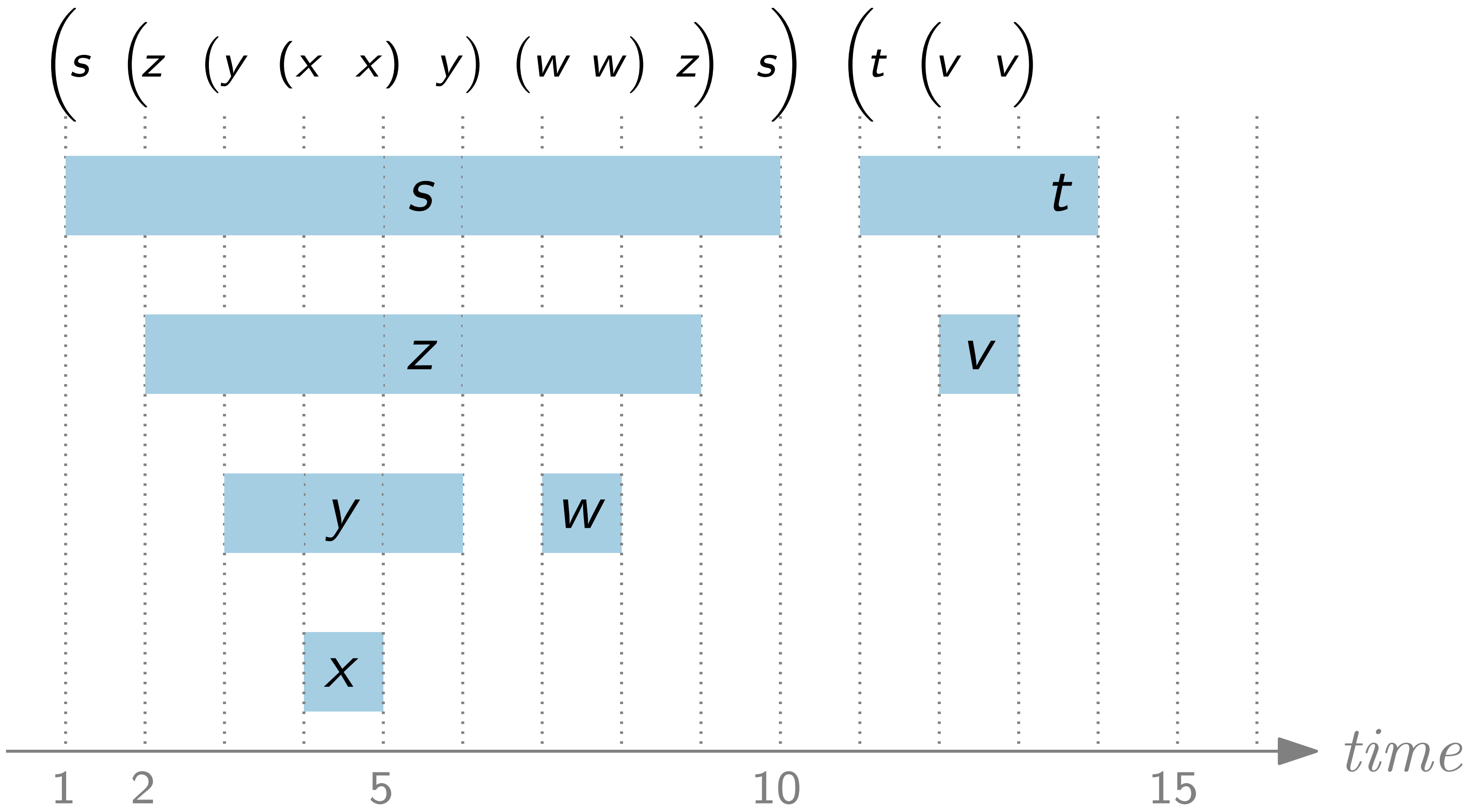
Tiefensuche – Eigenschaften



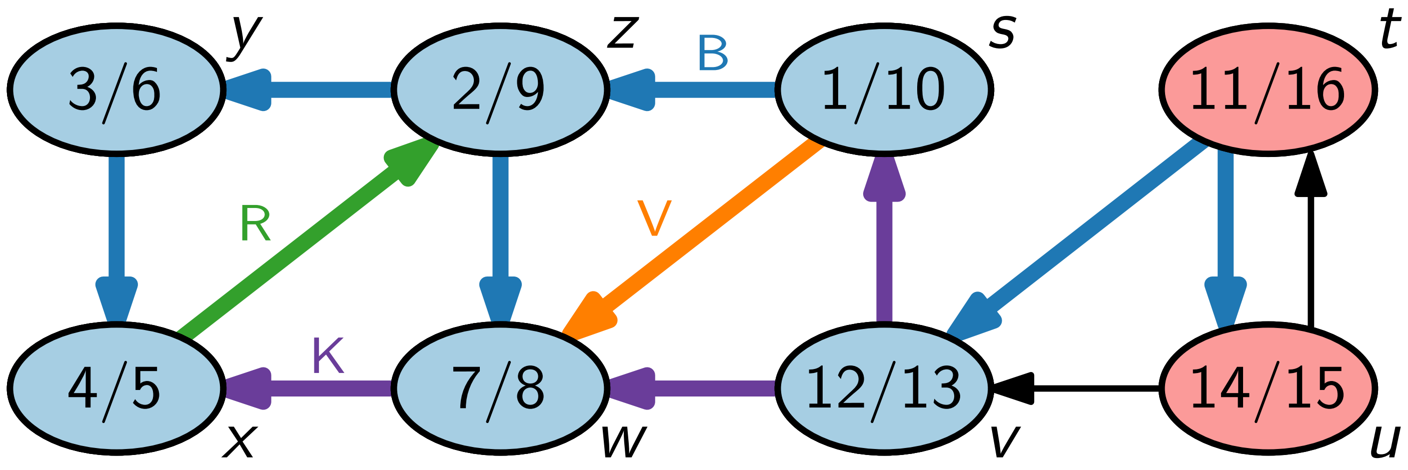


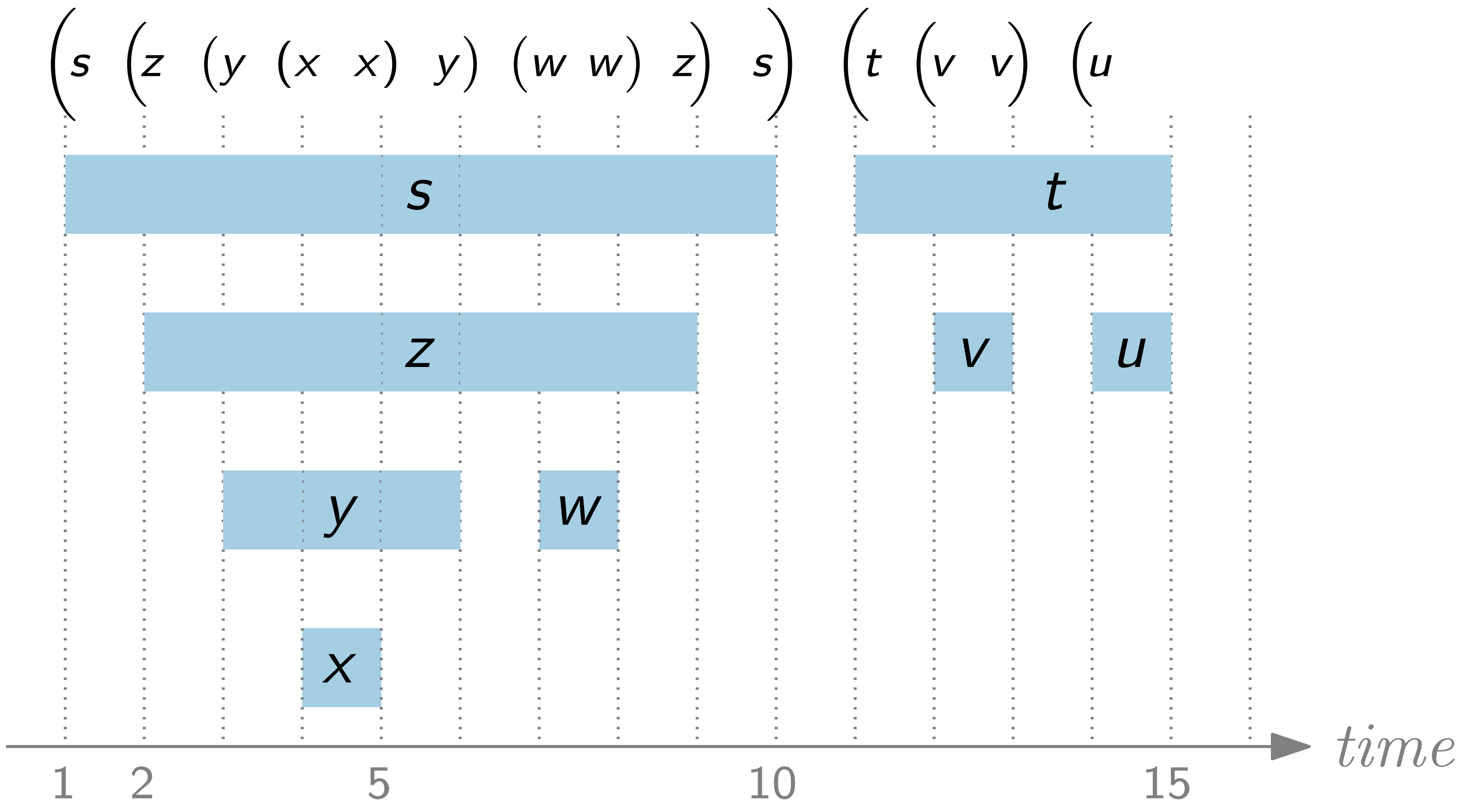
Tiefensuche – Eigenschaften



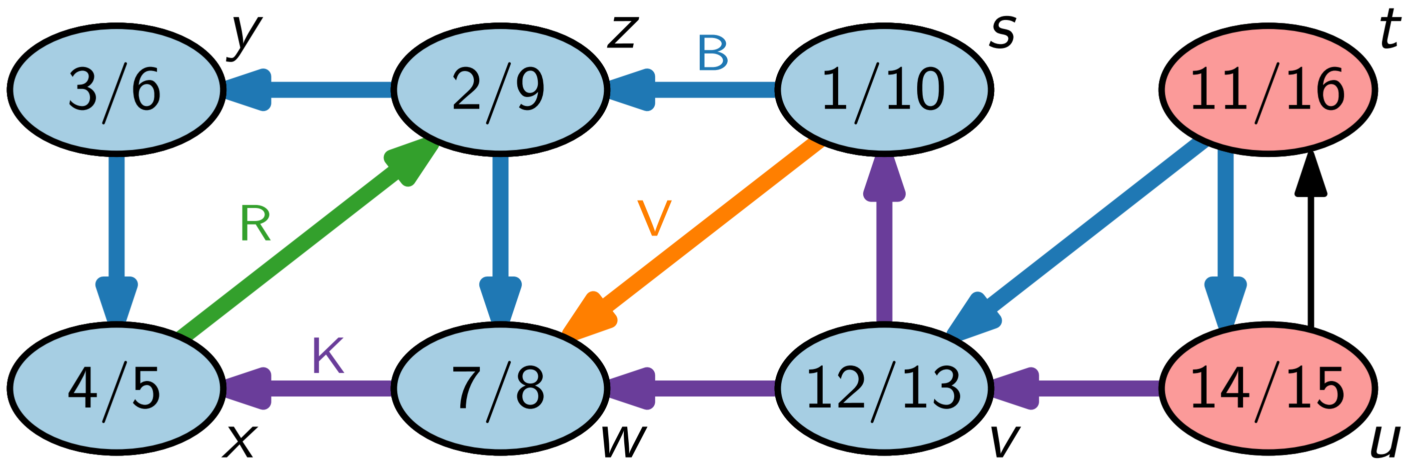


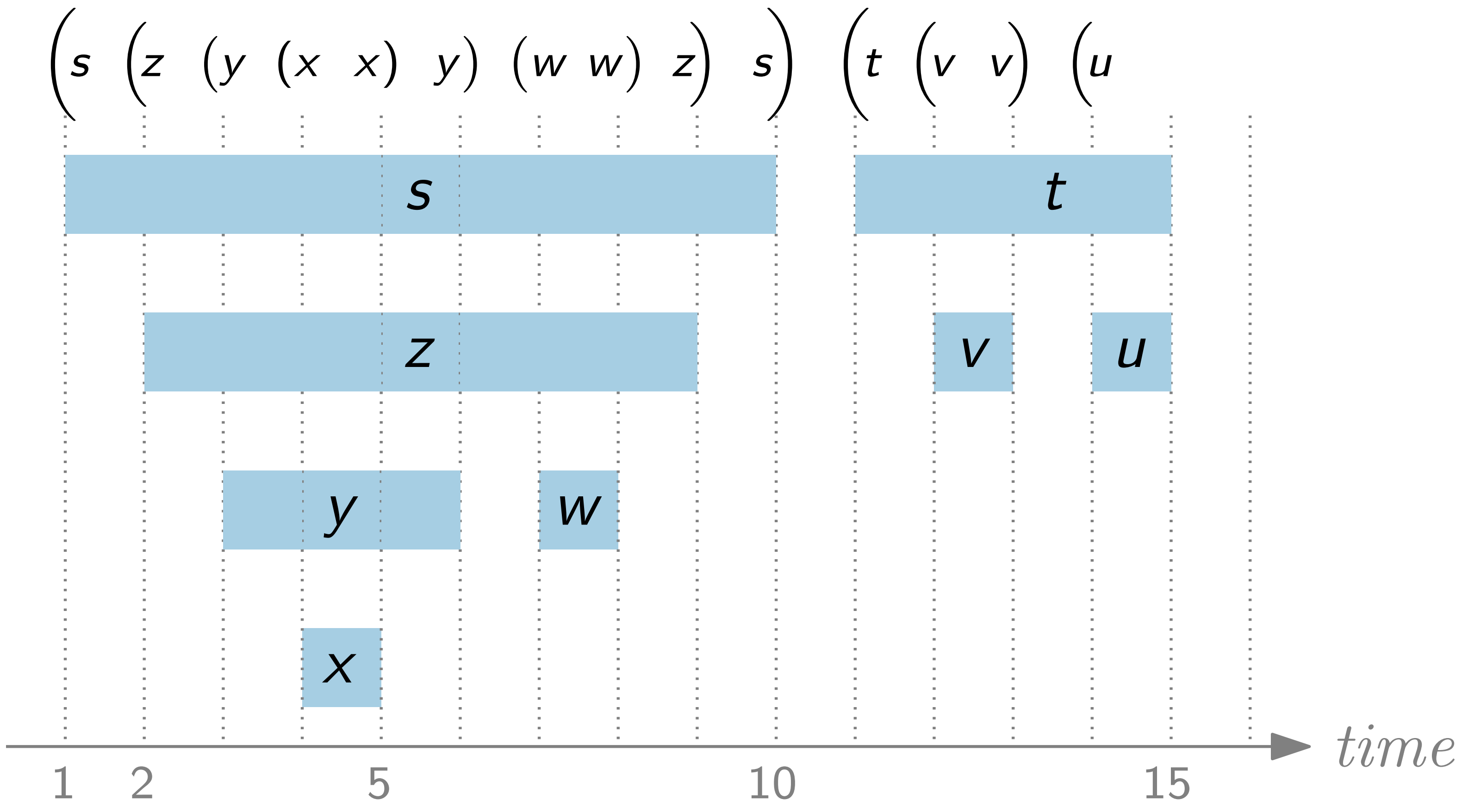
Tiefensuche – Eigenschaften



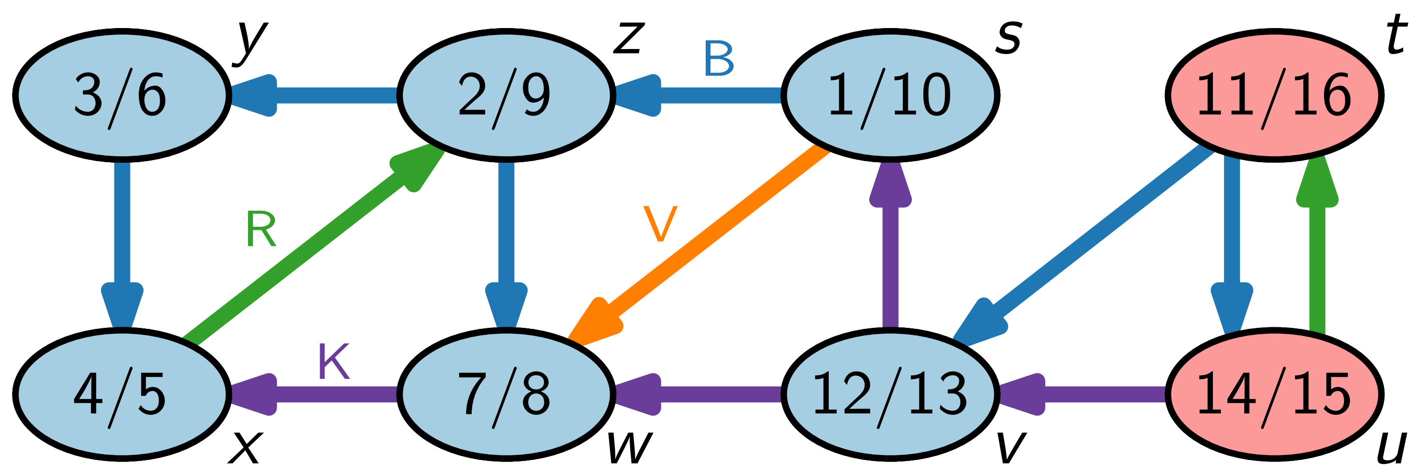


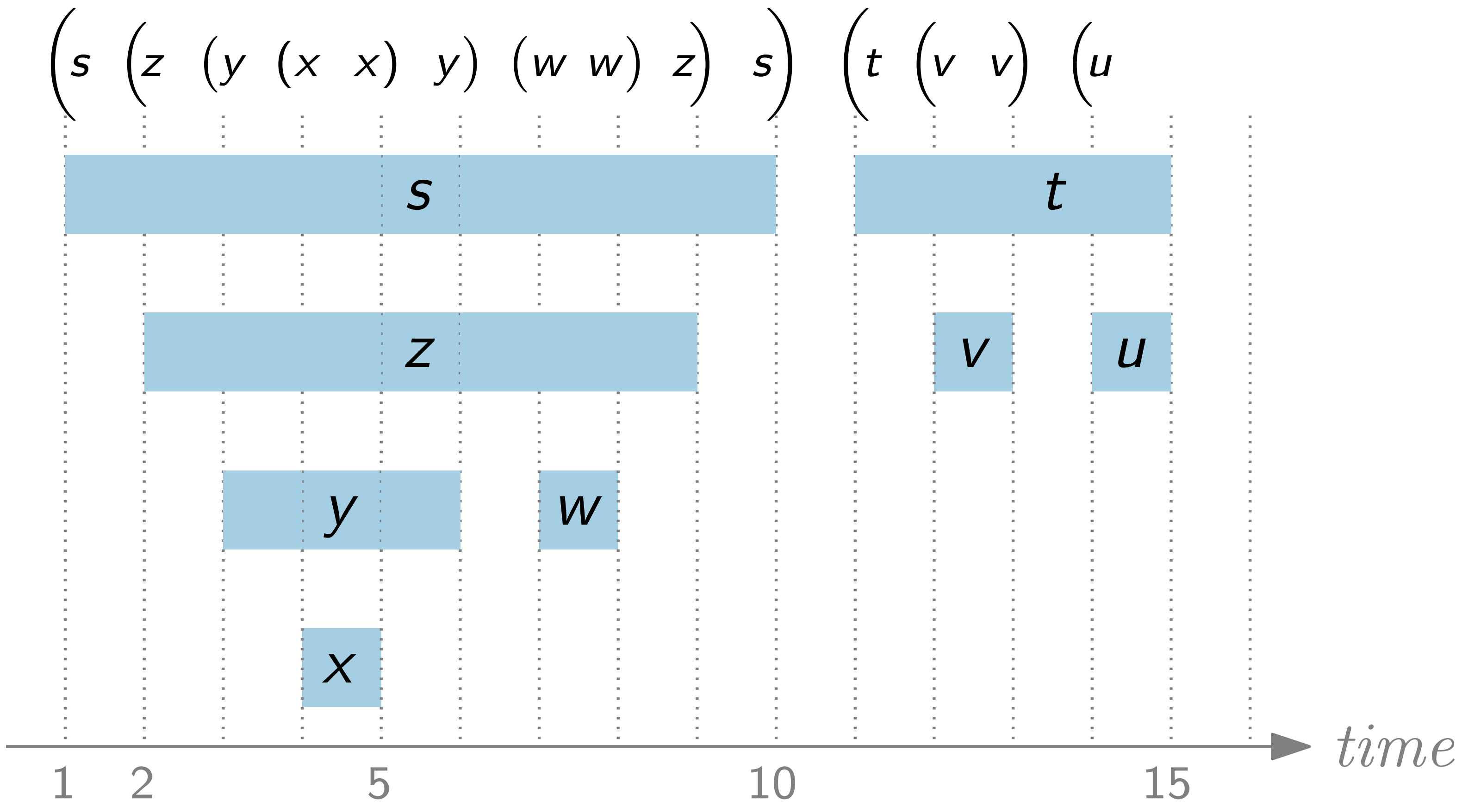
Tiefensuche – Eigenschaften



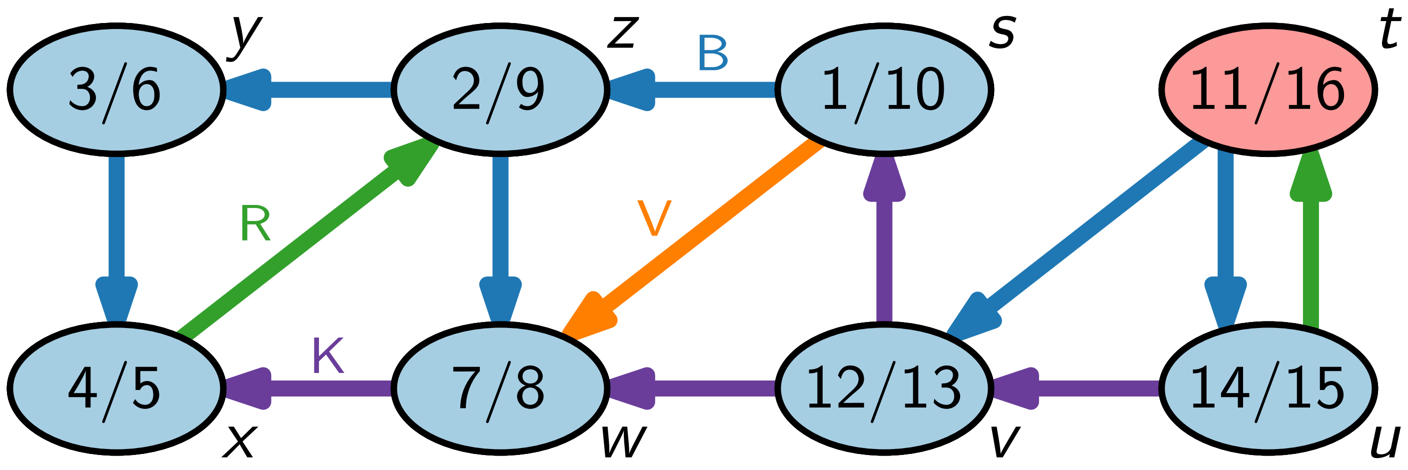


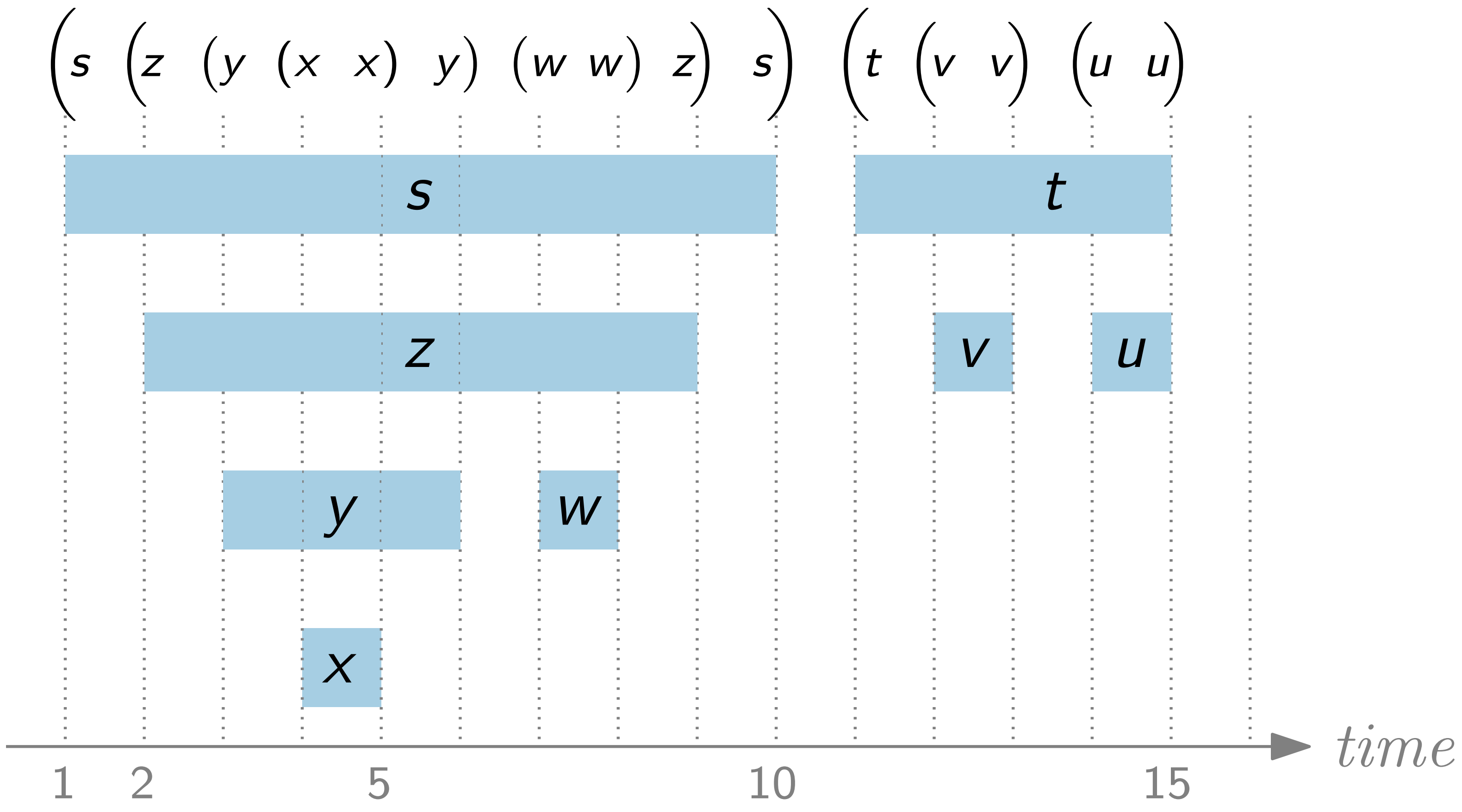
Tiefensuche – Eigenschaften



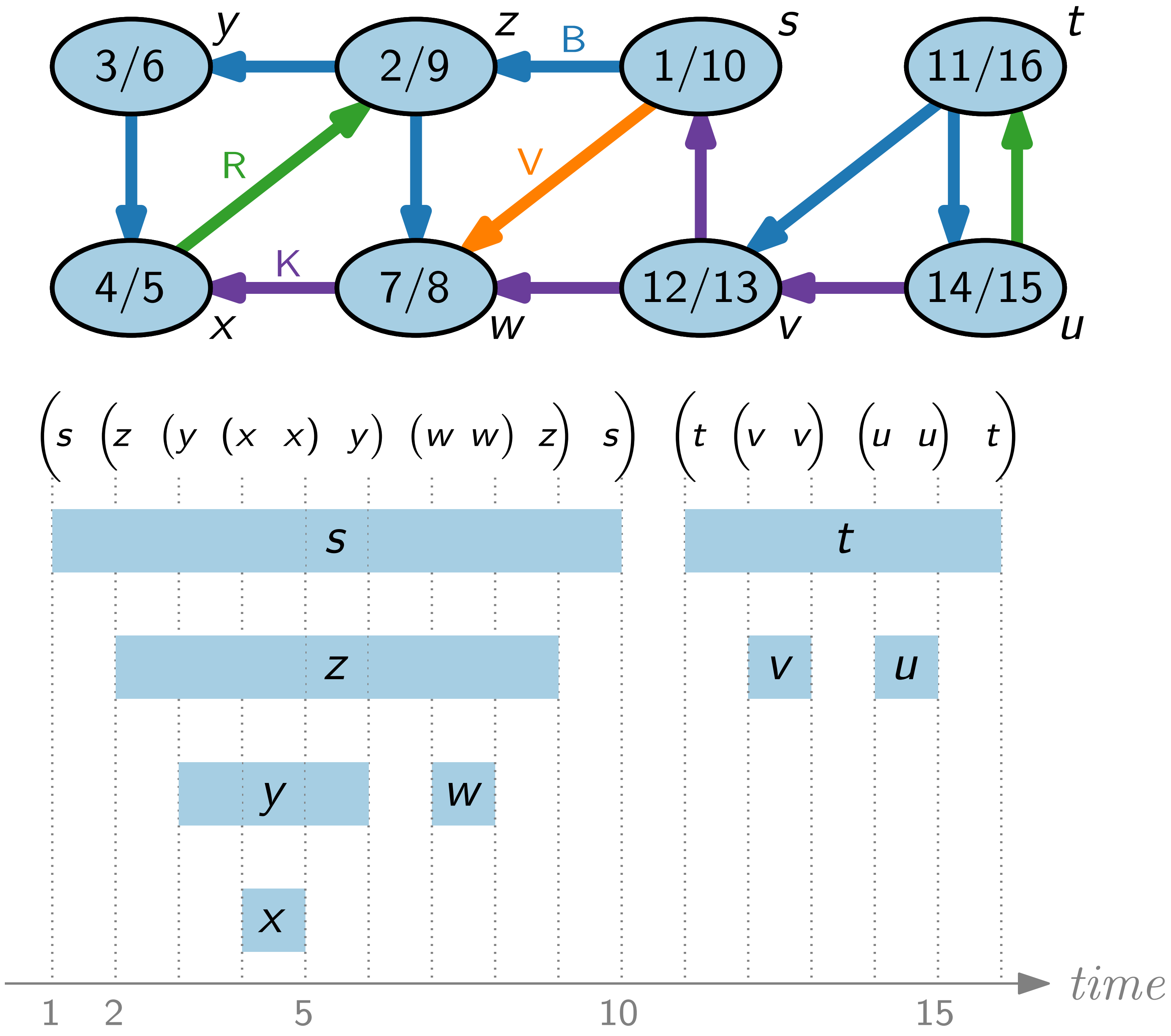


Tiefensuche – Eigenschaften

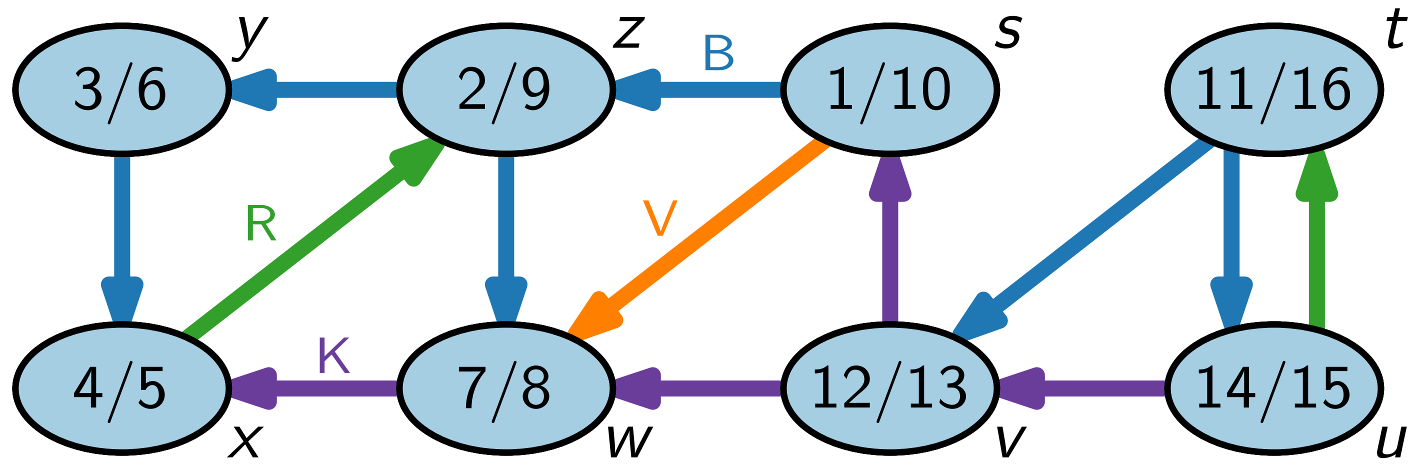


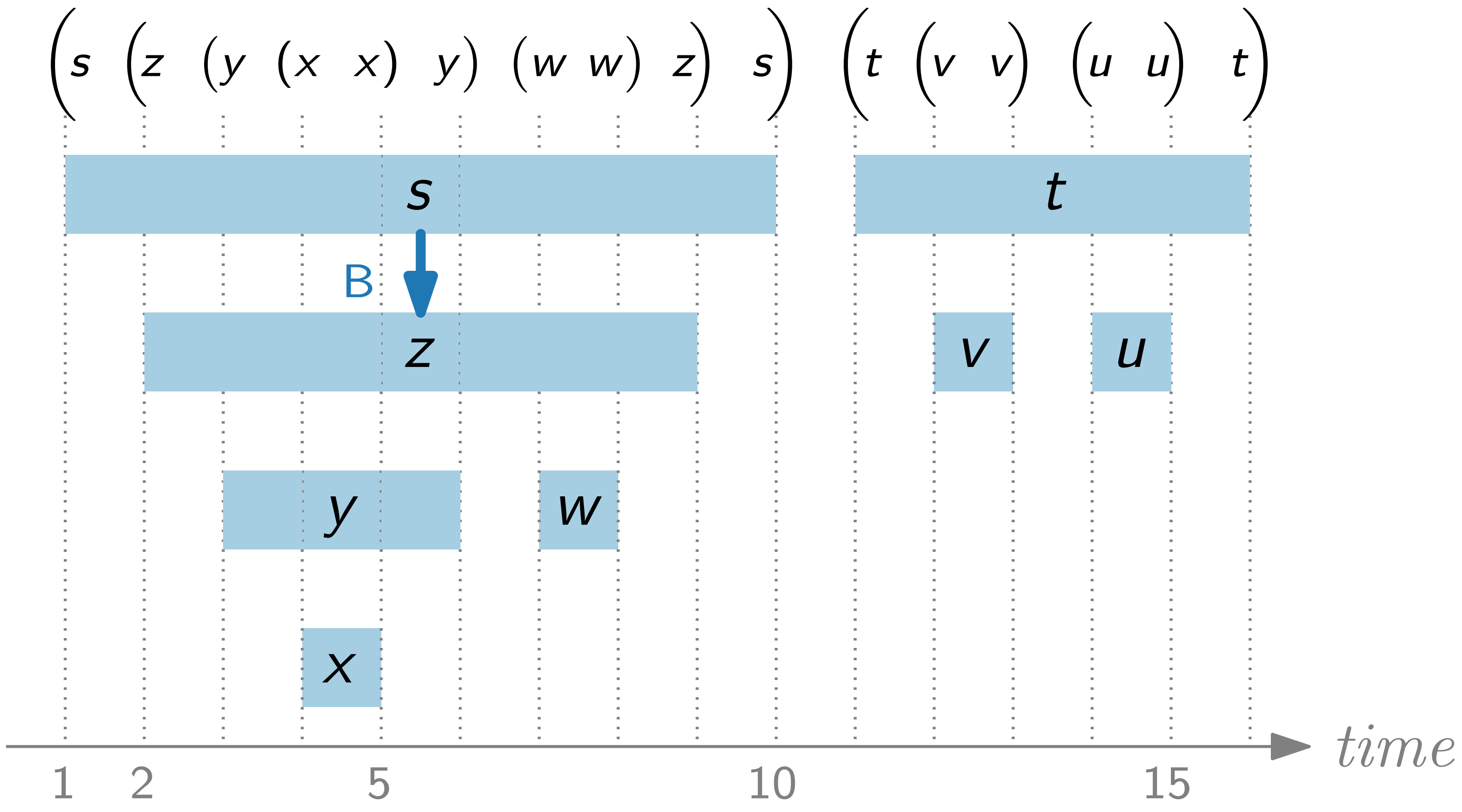


Tiefensuche – Eigenschaften

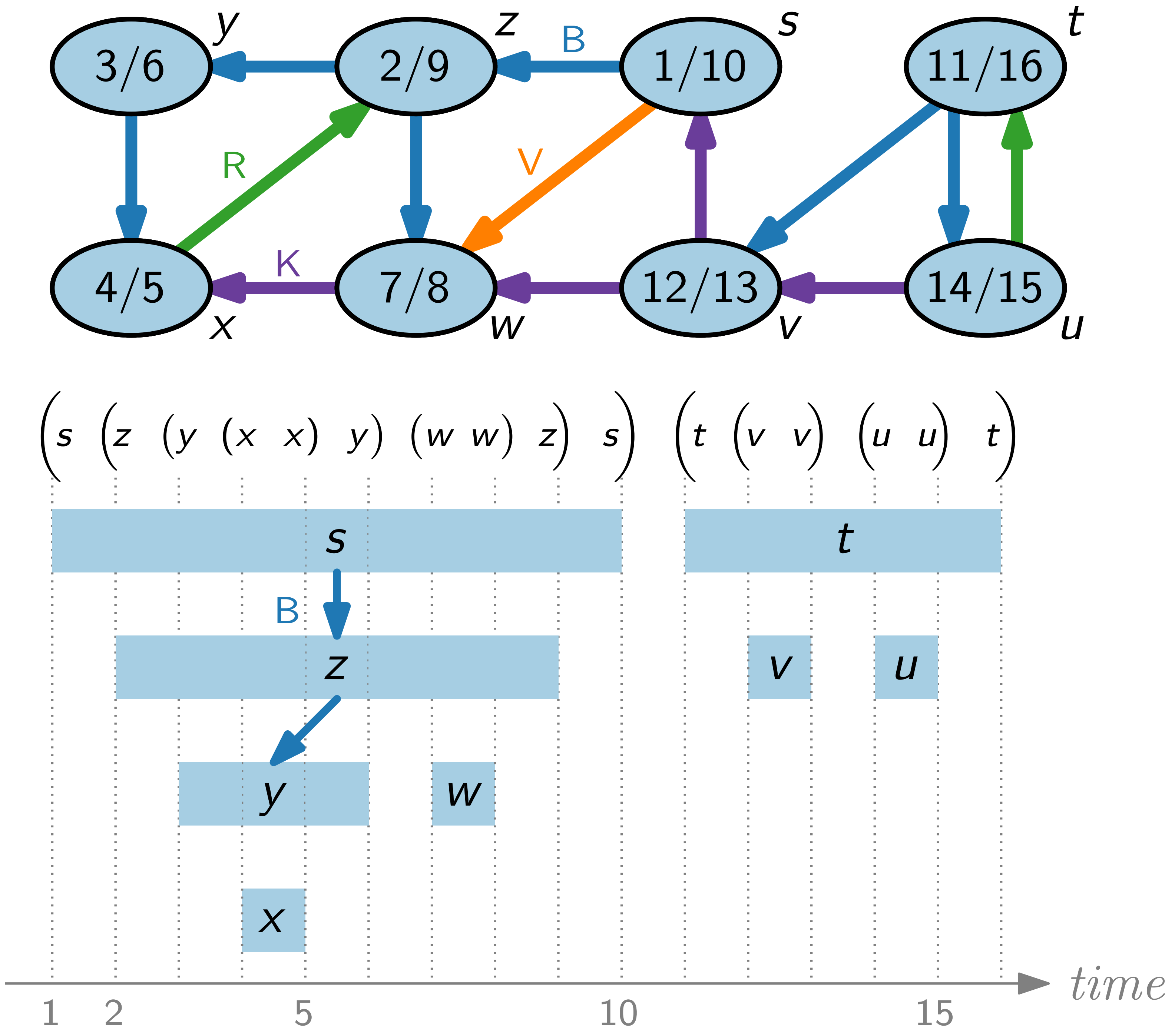


Tiefensuche – Eigenschaften

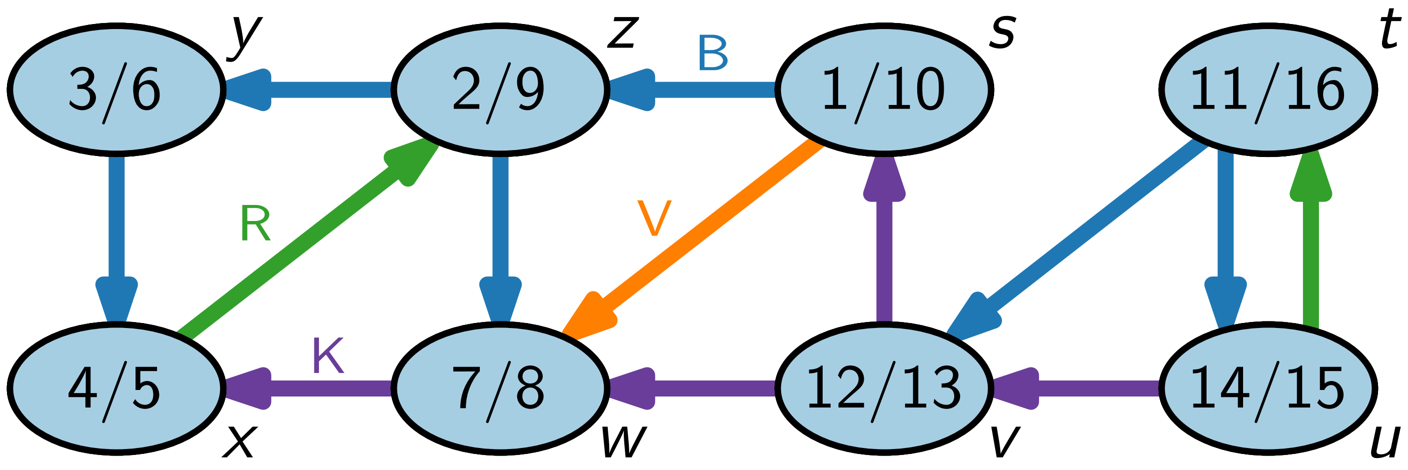


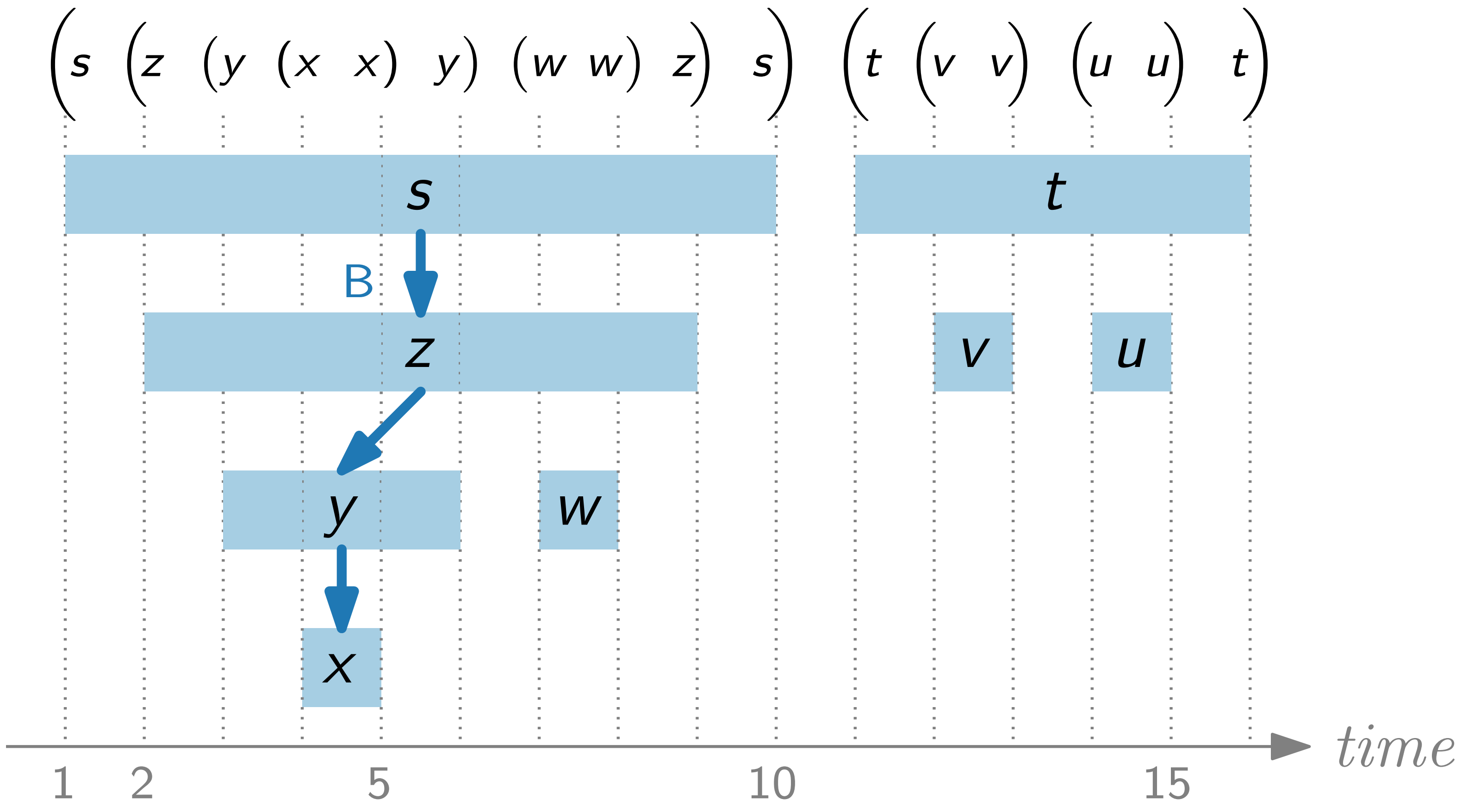


Tiefensuche – Eigenschaften

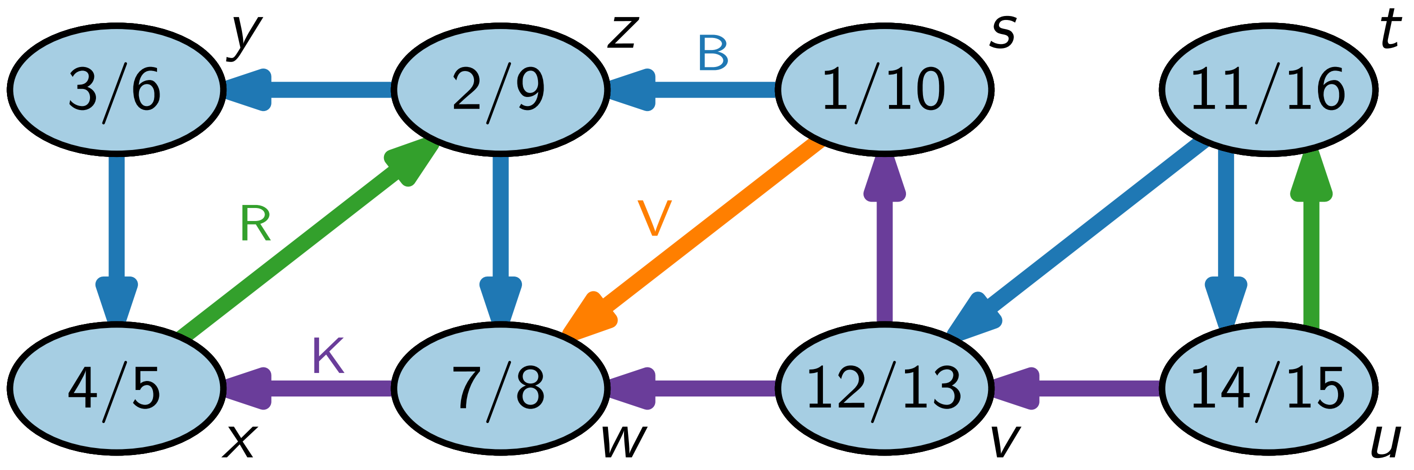


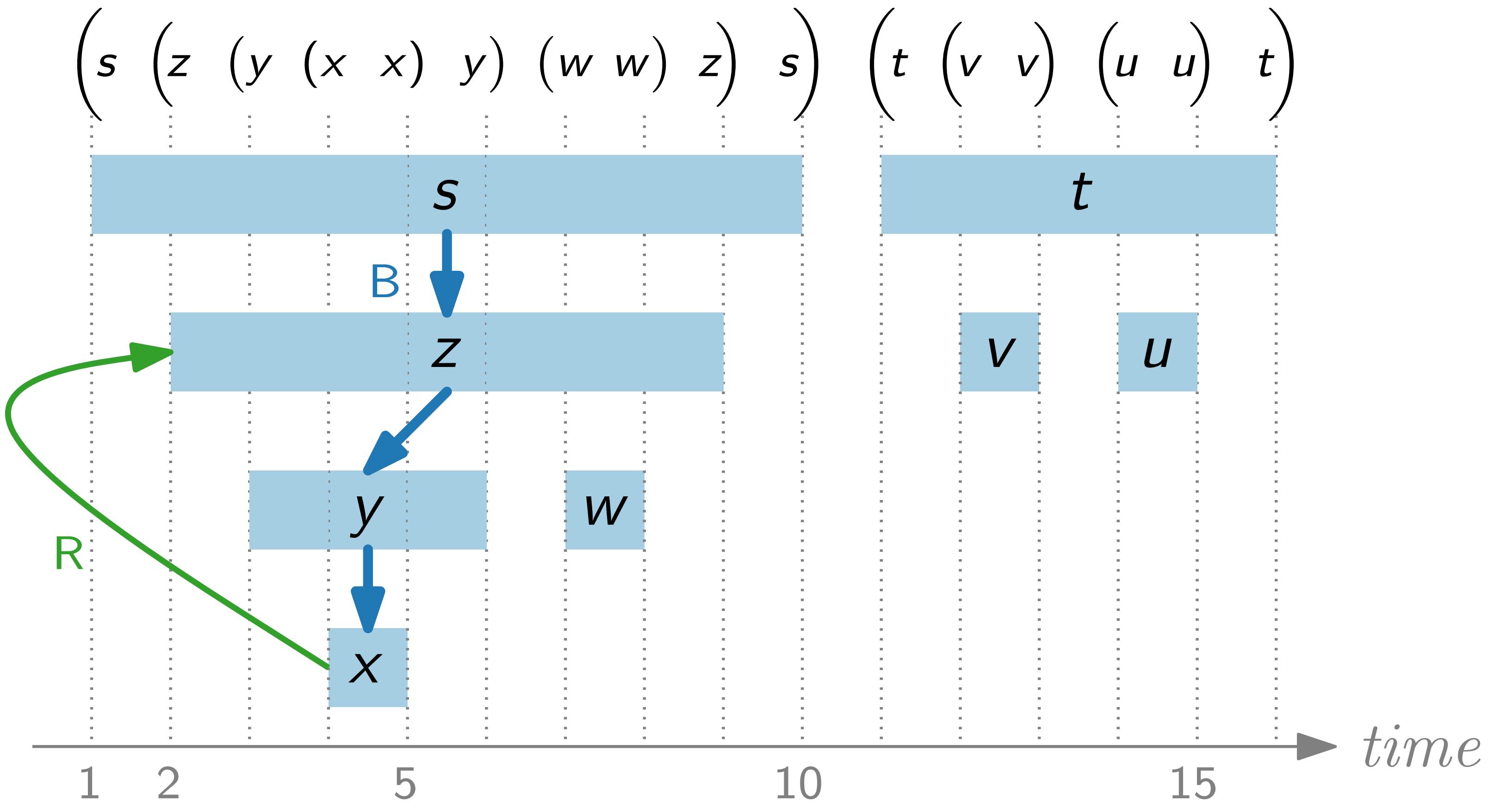
Tiefensuche – Eigenschaften



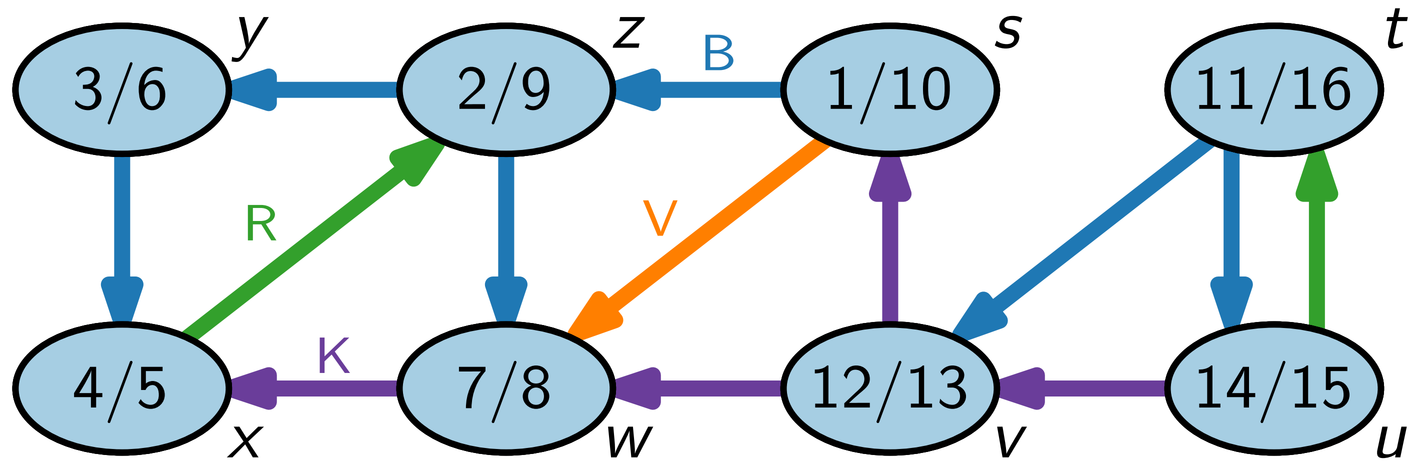


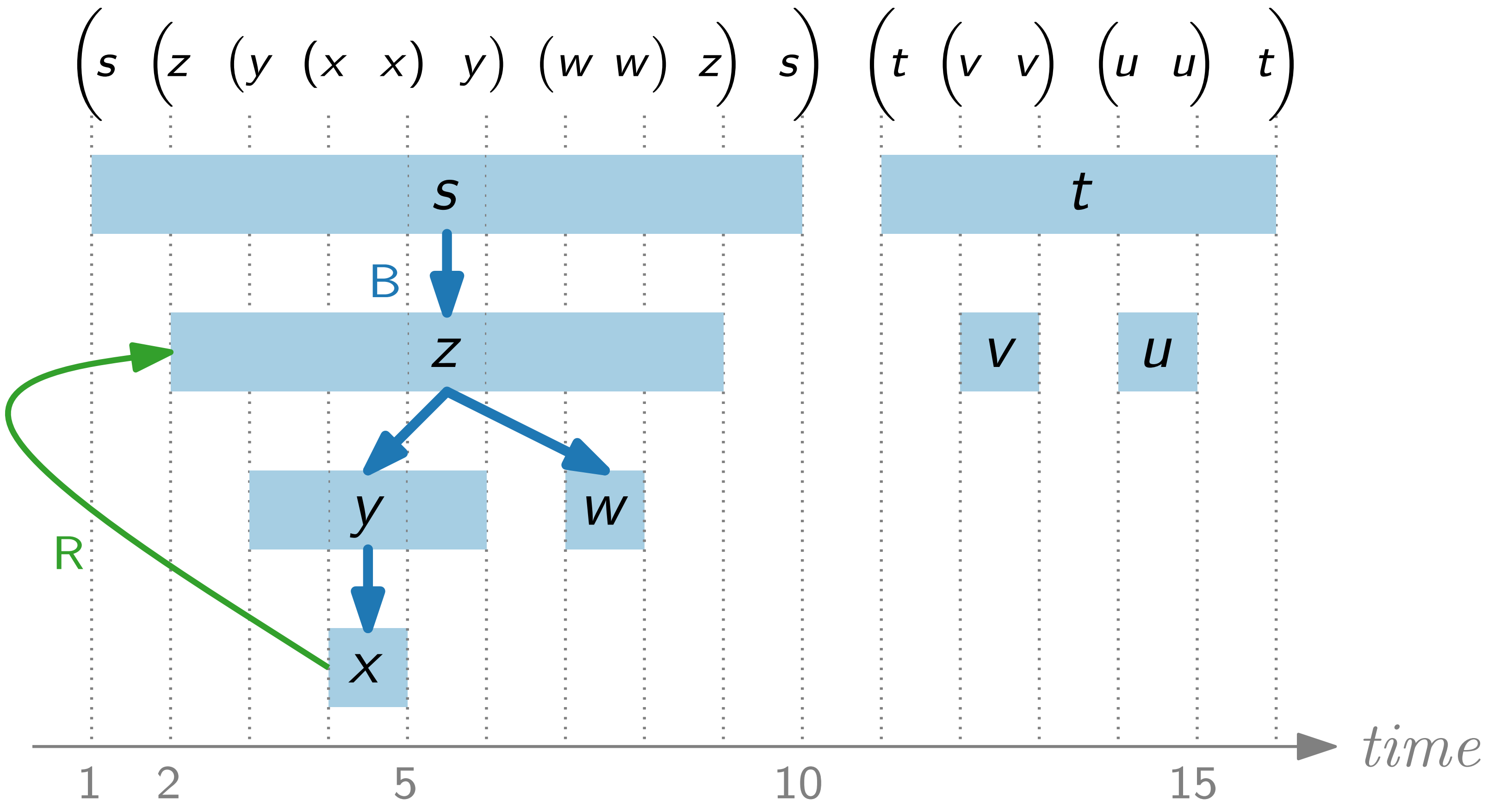
Tiefensuche – Eigenschaften



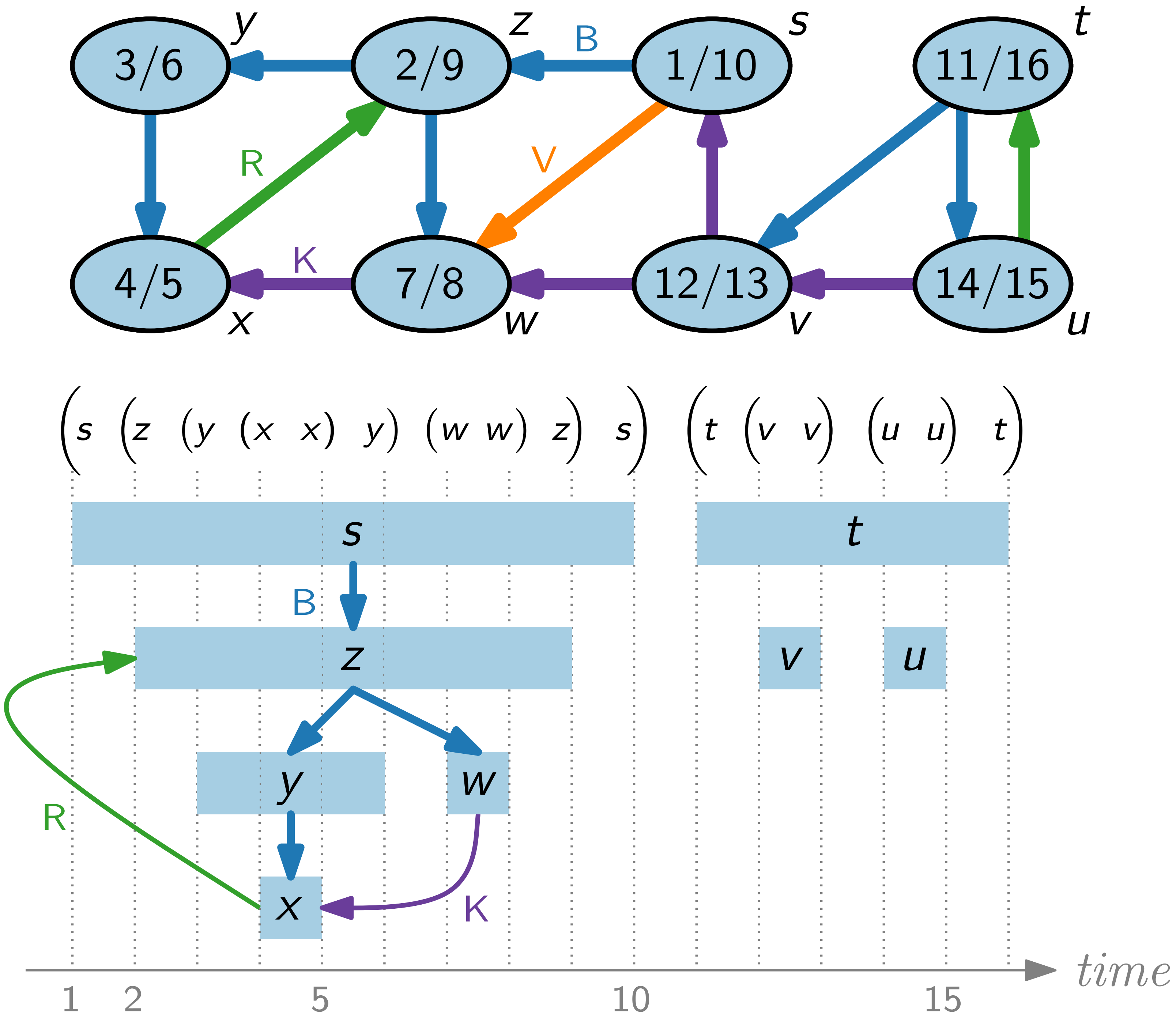


Tiefensuche – Eigenschaften

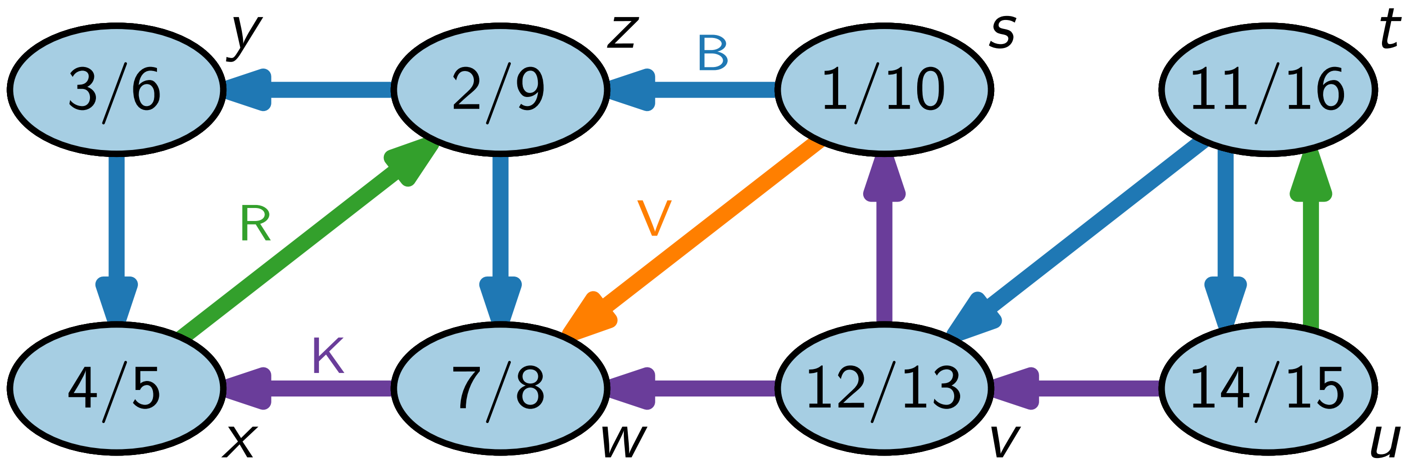


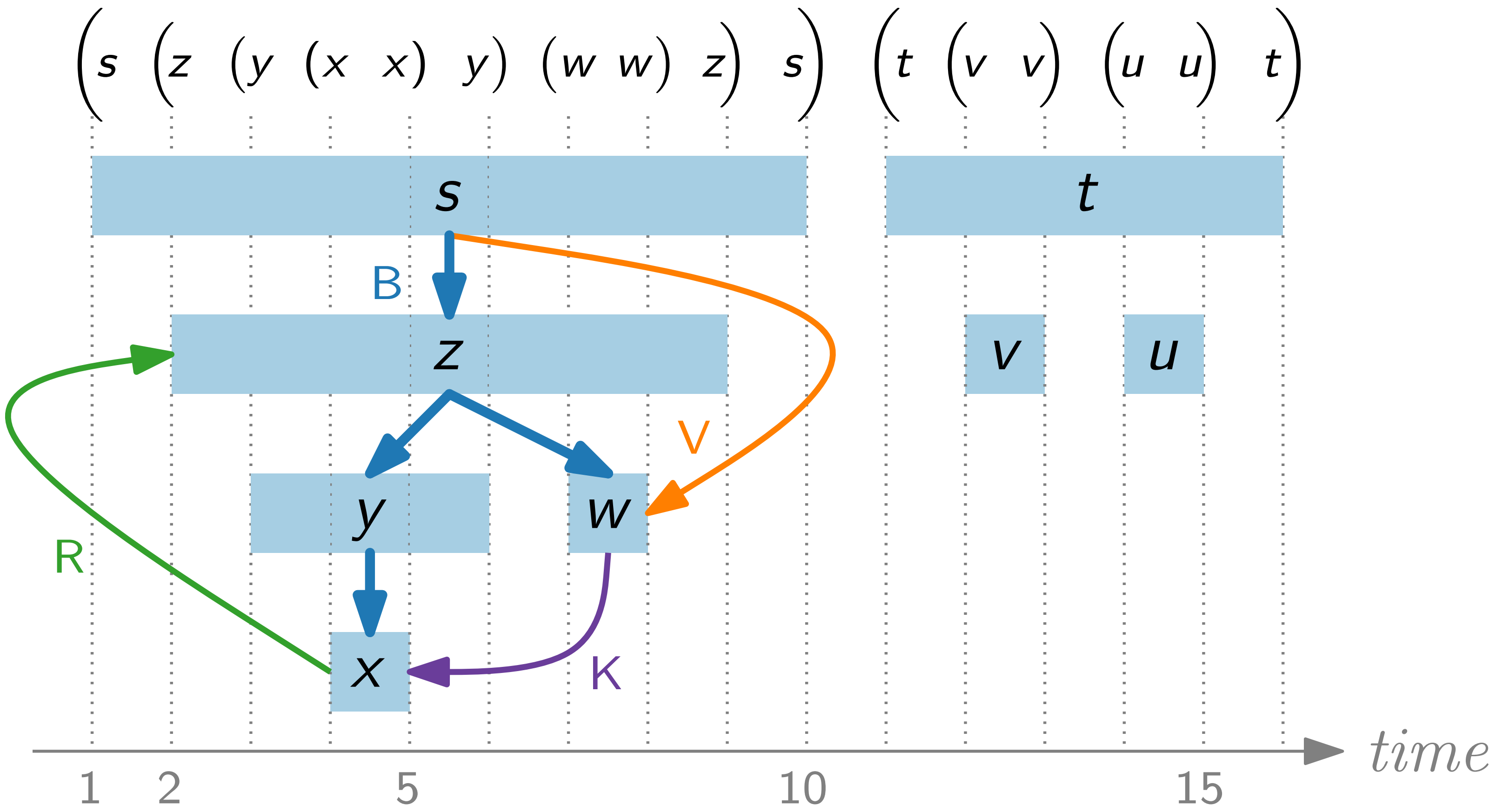


Tiefensuche – Eigenschaften

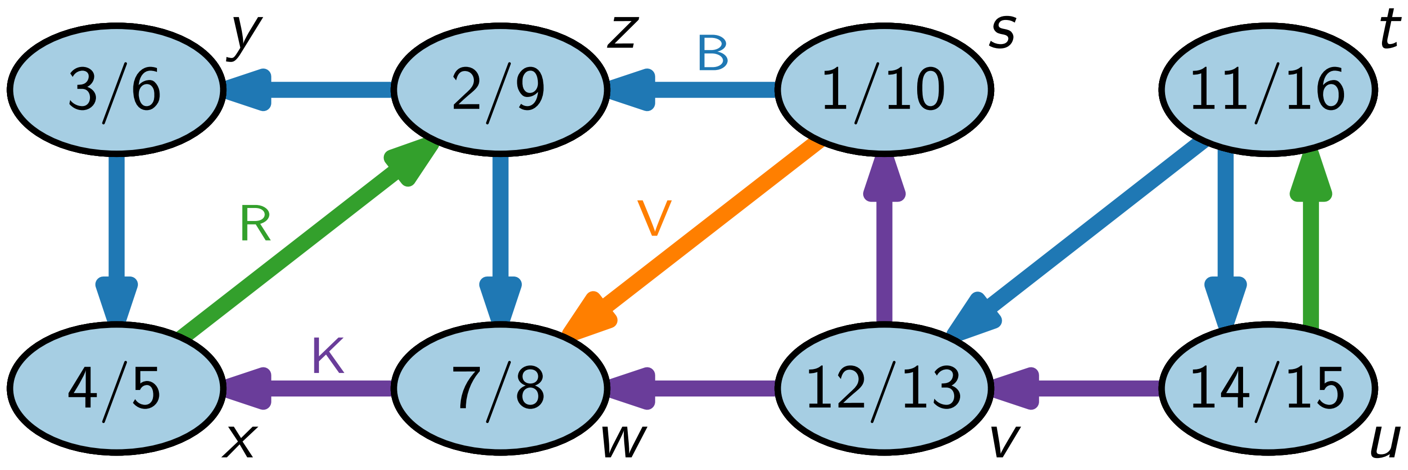


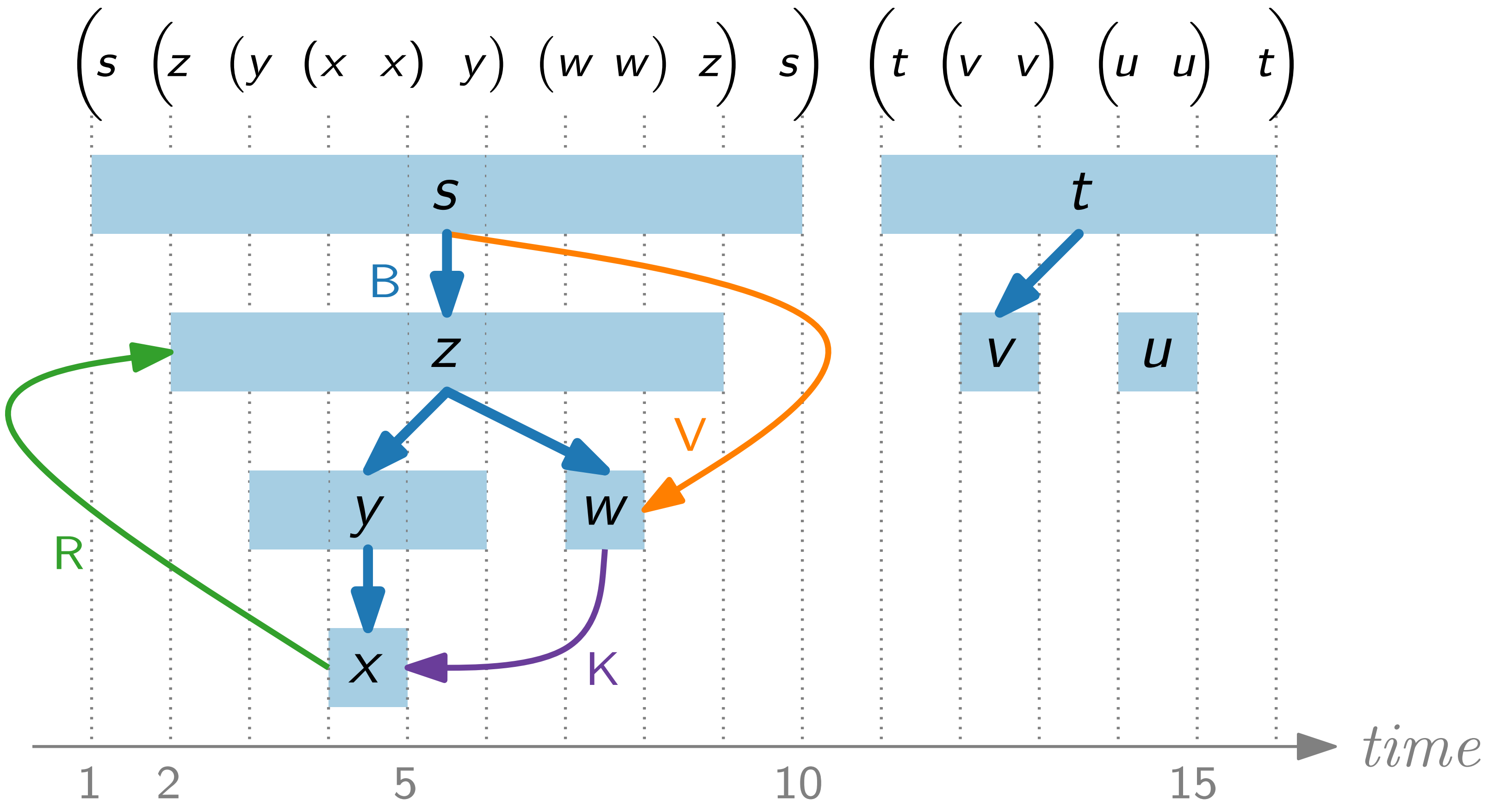
Tiefensuche – Eigenschaften



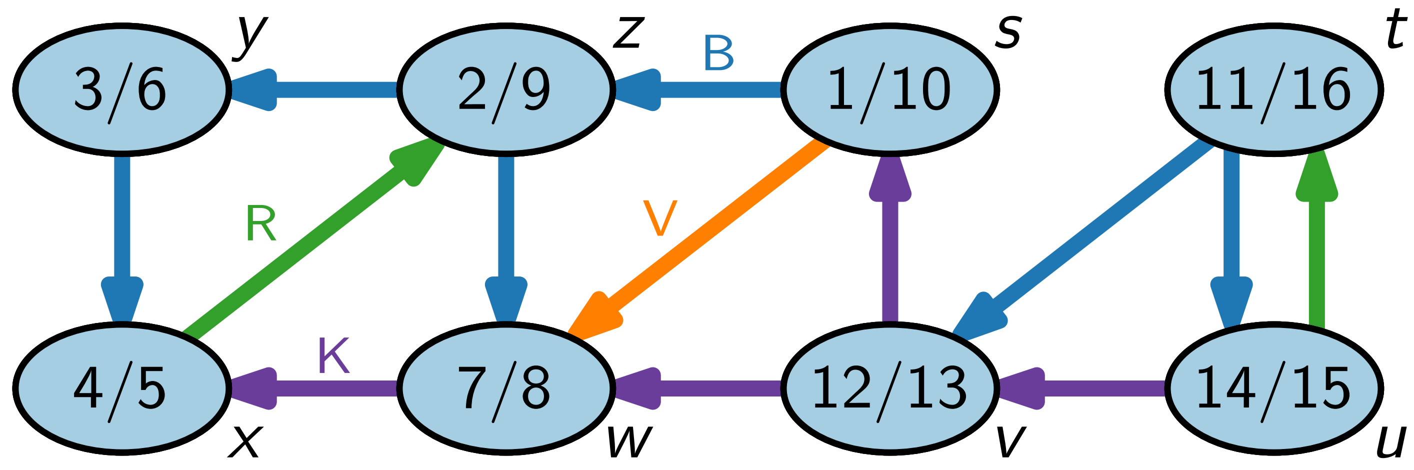


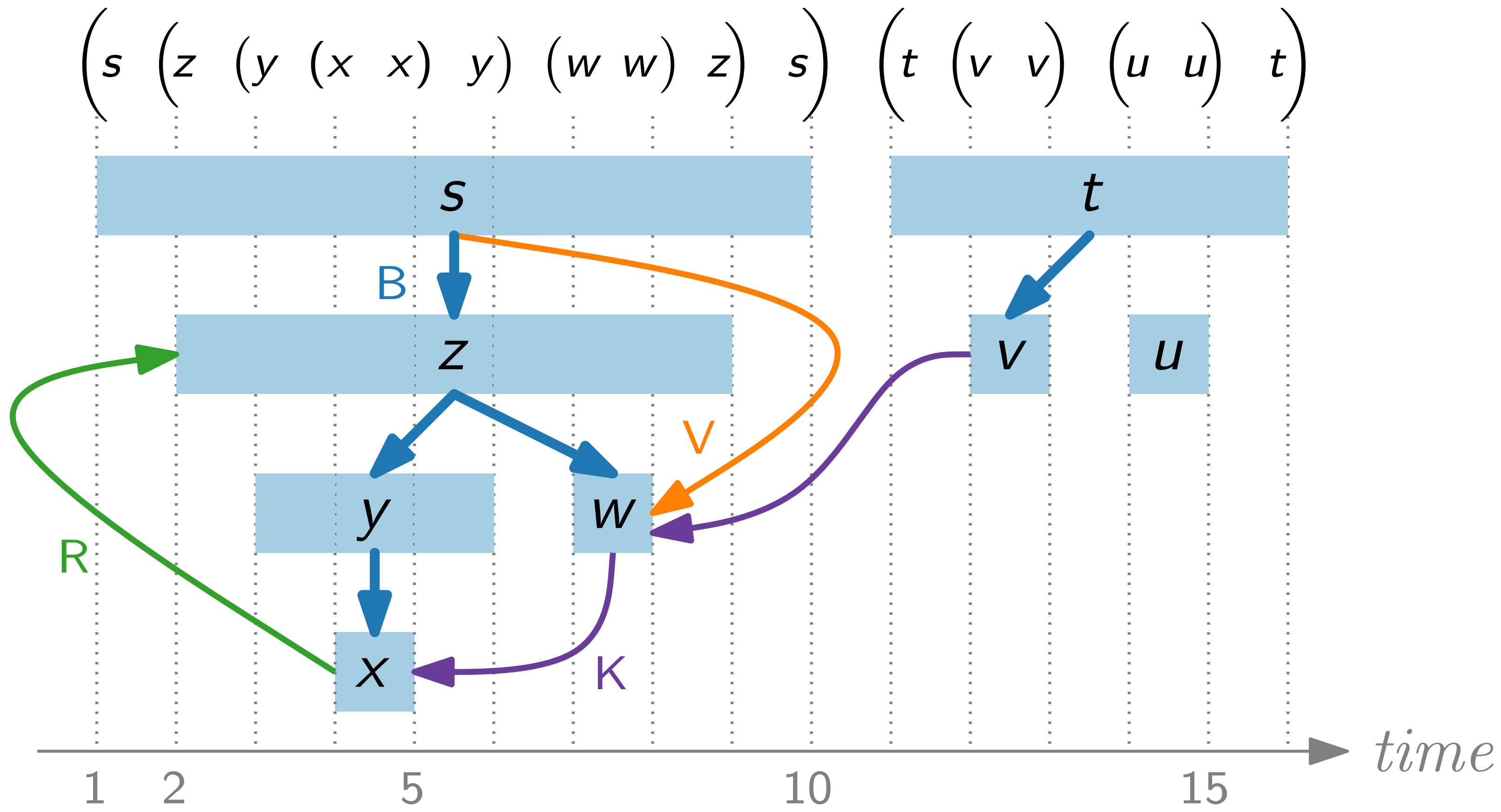
Tiefensuche – Eigenschaften



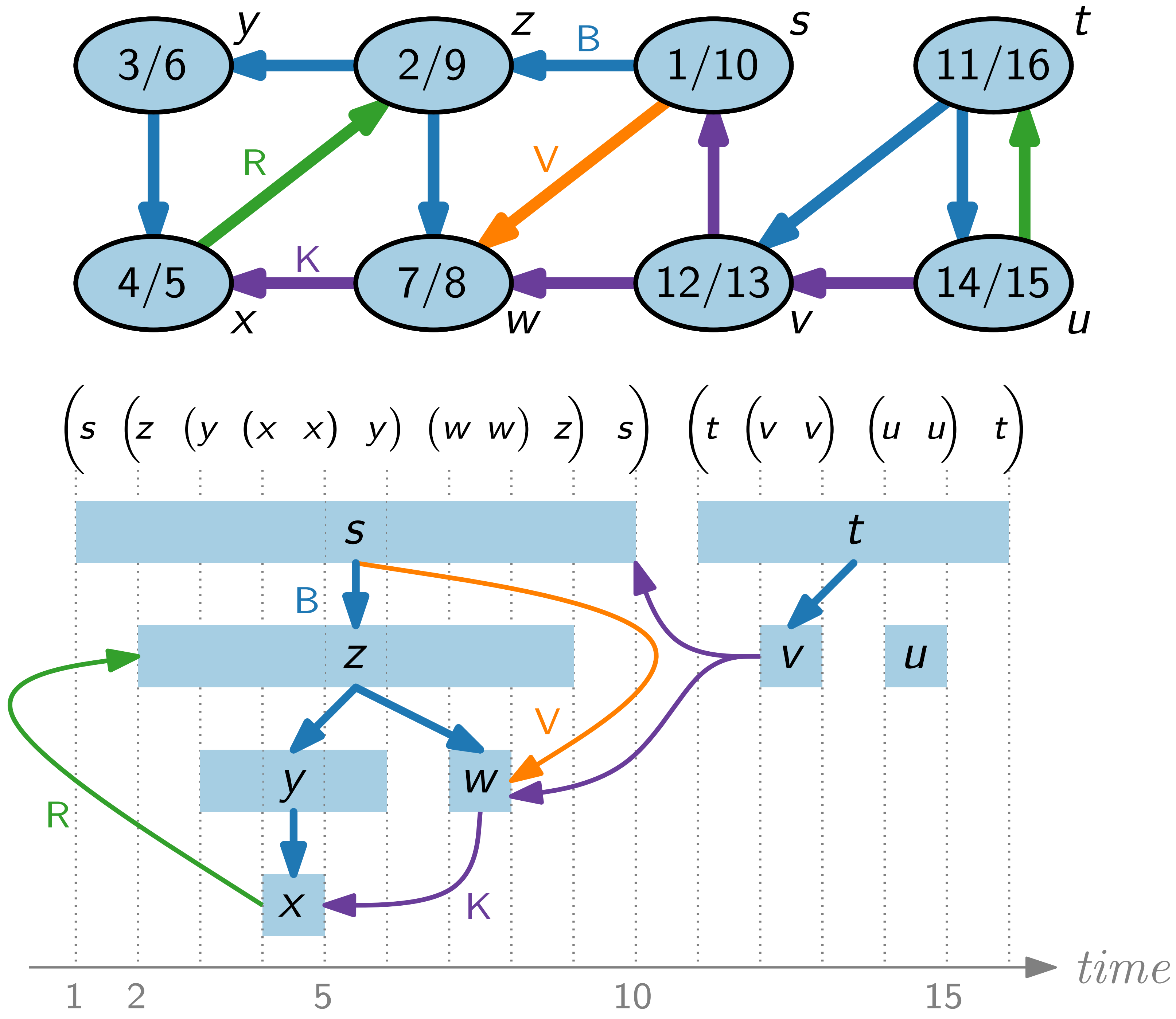


Tiefensuche – Eigenschaften

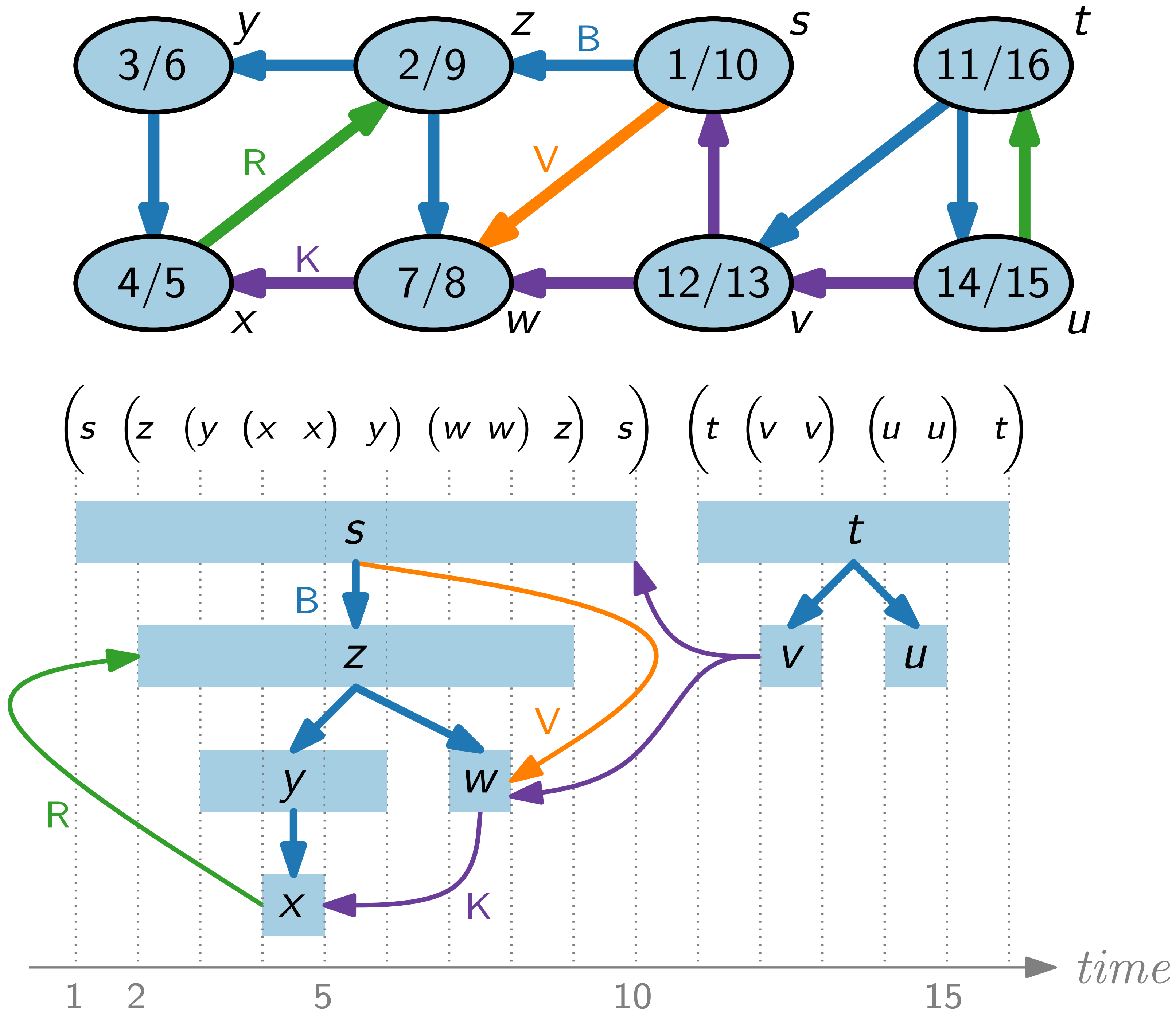




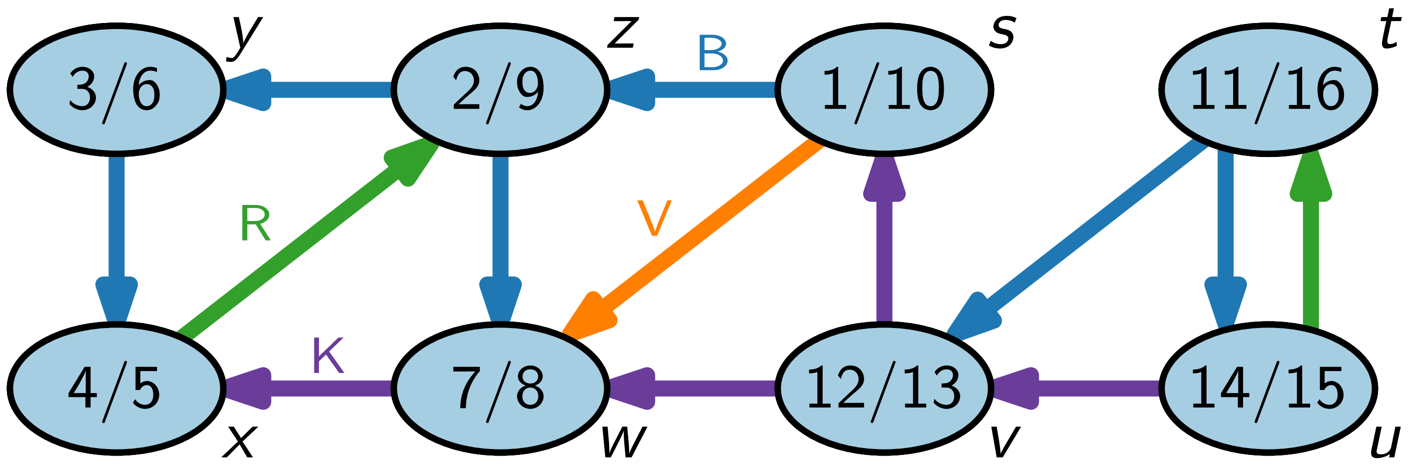
Tiefensuche – Eigenschaften

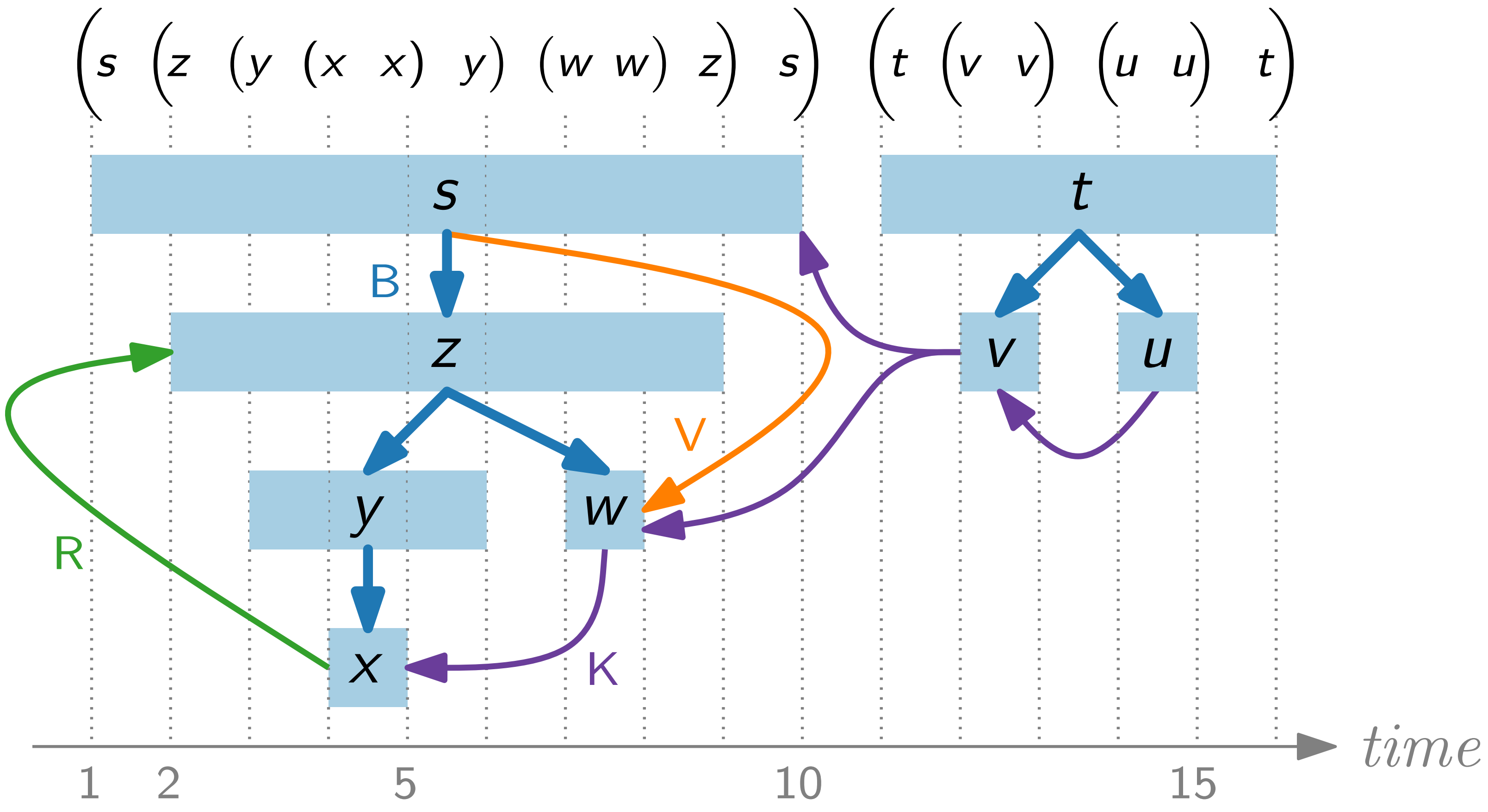


Tiefensuche – Eigenschaften

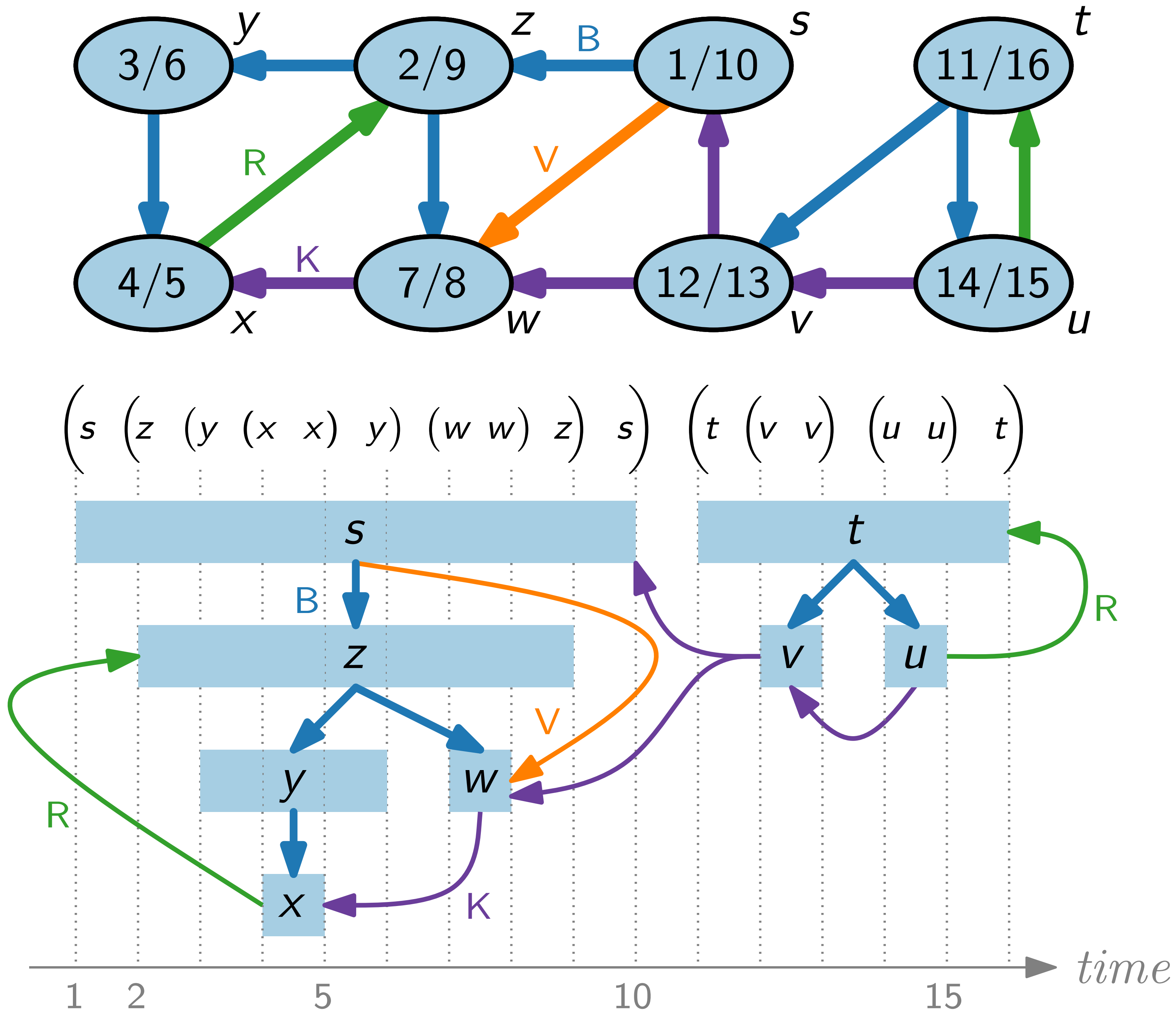


Tiefensuche – Eigenschaften





Tiefensuche – Eigenschaften



Tiefensuche – Analyse

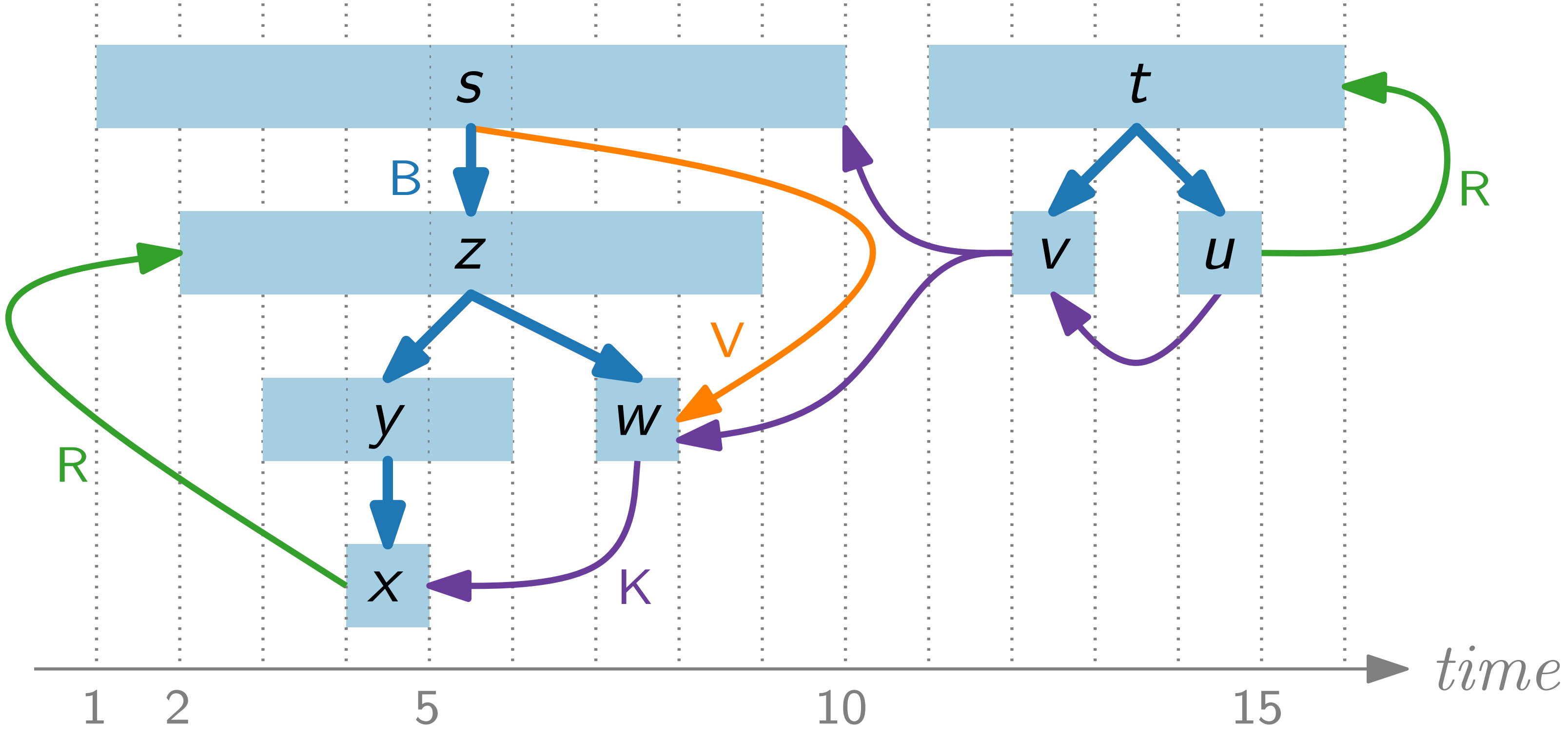
Satz. (Klammerntheorem)

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

(i)

(ii)

(iii)



Tiefensuche – Analyse

Satz. (Klammerntheorem)

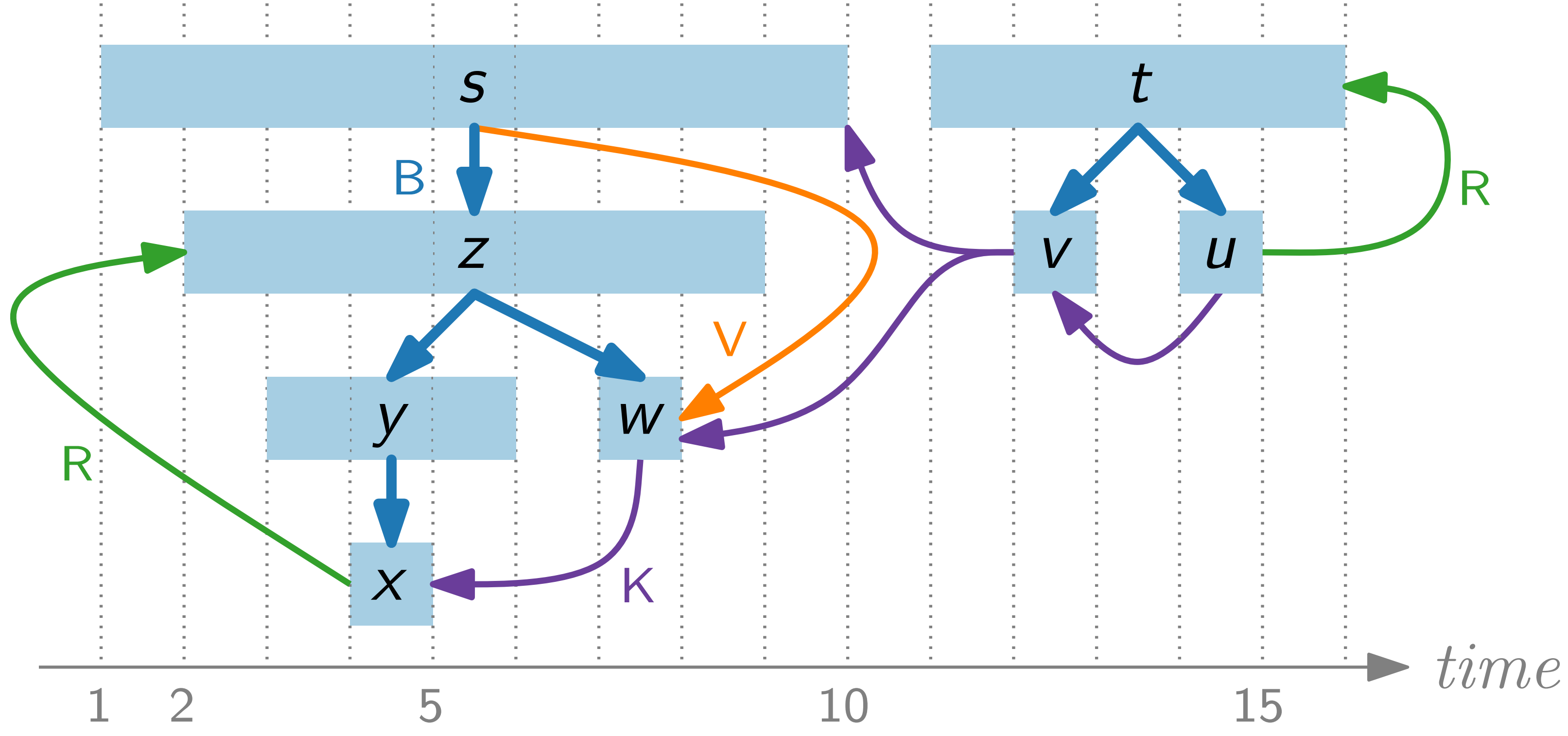
d.h. für jedes Paar $\{u, v\}$ von Knoten (mit $u \neq v$)

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

(i)

(ii)

(iii)



Tiefensuche – Analyse

Satz. (Klammerntheorem)

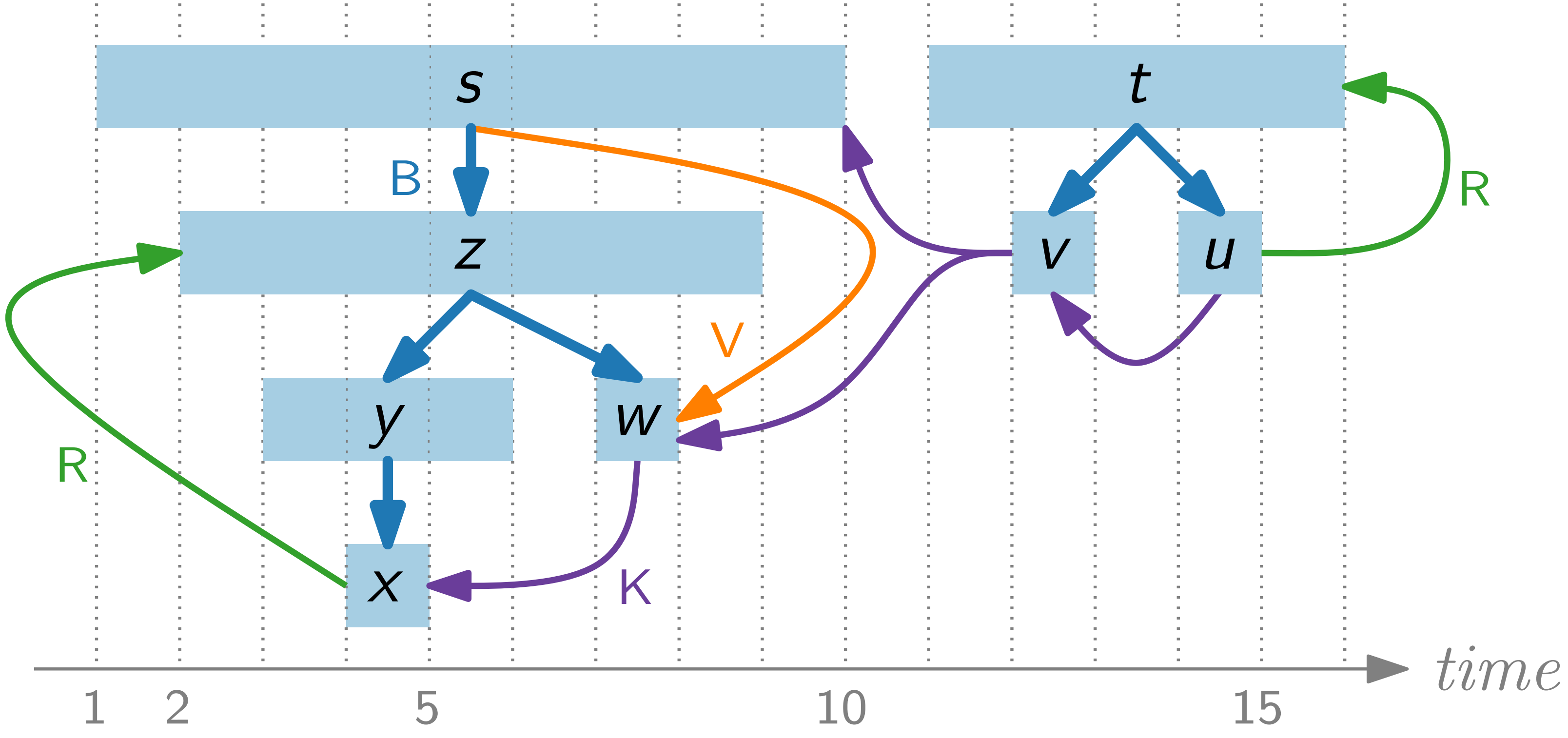
d.h. für jedes Paar $\{u, v\}$ von Knoten (mit $u \neq v$)

Nach 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. (Klammerntheorem)

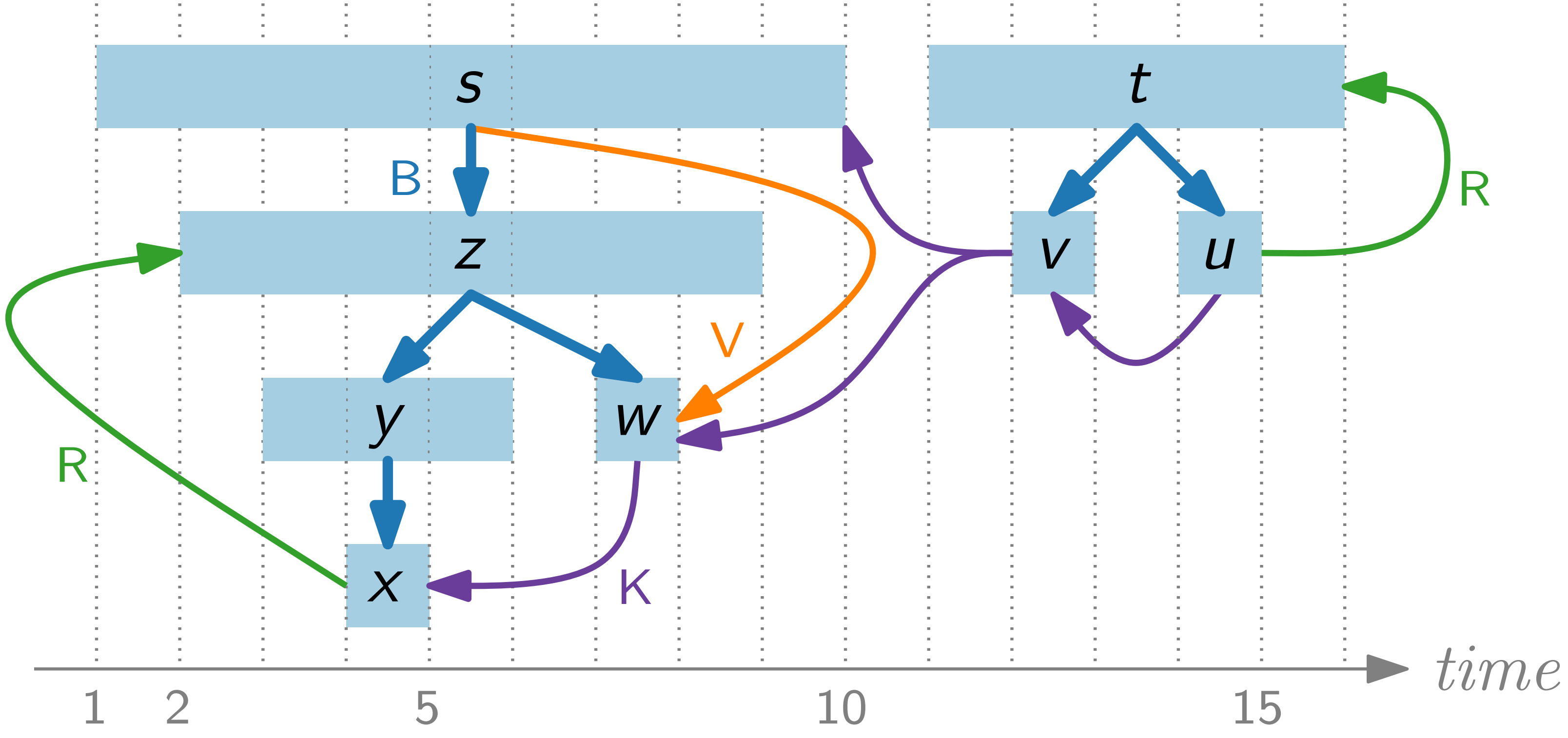
d.h. für jedes Paar $\{u, v\}$ von Knoten (mit $u \neq v$)

Nach 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. (Klammerntheorem)

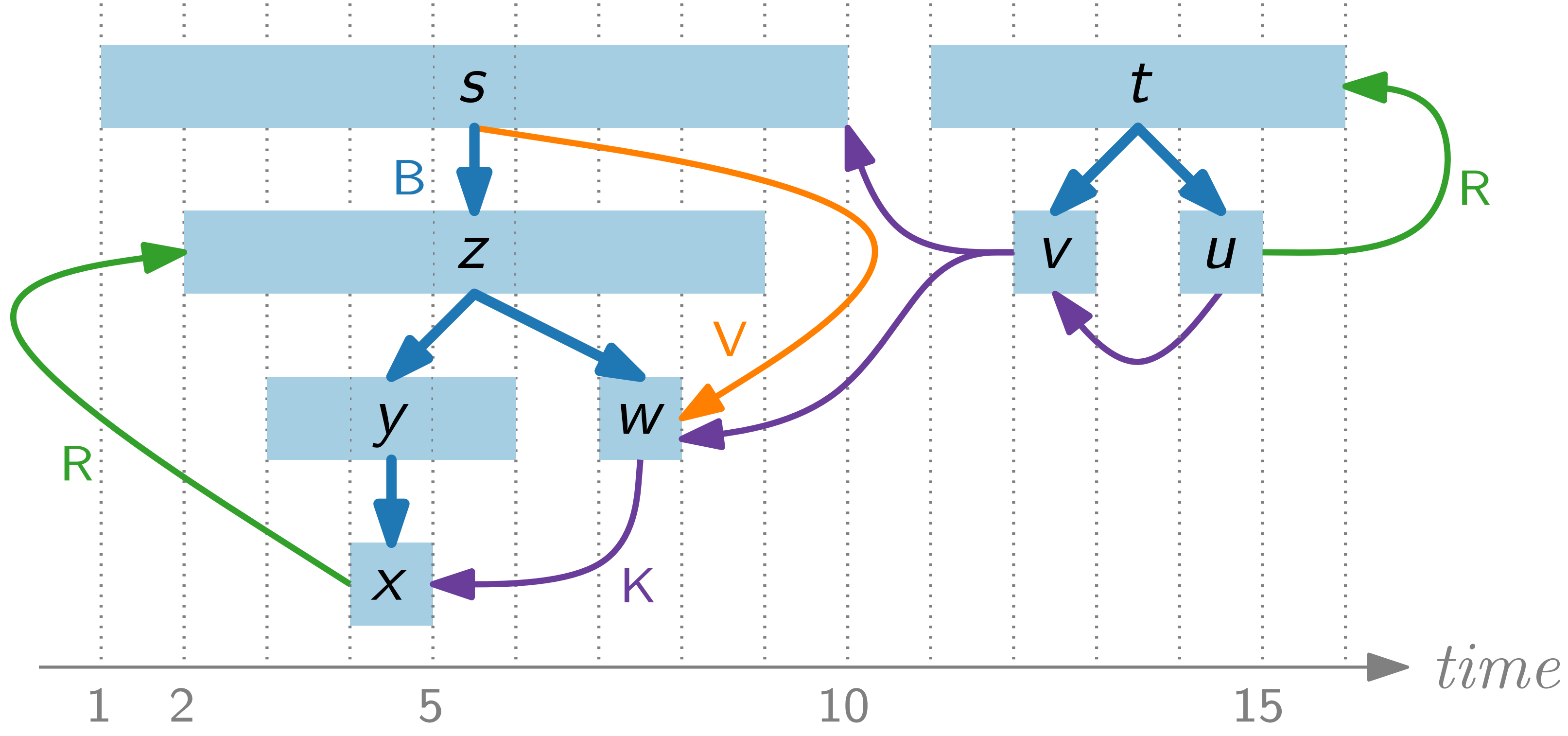
d.h. für jedes Paar $\{u, v\}$ von Knoten (mit $u \neq v$)

Nach 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. (Klammerntheorem)

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

Nach 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.

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

Beweis. Wir betrachten zwei Fälle.

Tiefensuche – Analyse

Satz. (Klammerntheorem)

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

Nach 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.

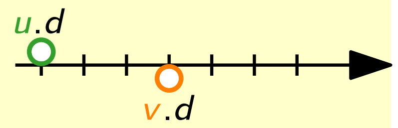
(ii) $[u.d, u.f] \subset [v.d, v.f]$ und Baumkanten enthalten v-u-Weg.

(iii) Wie (ii), nur umgekehrt.

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

Beweis. Wir betrachten zwei Fälle.

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



Tiefensuche – Analyse

Satz. (Klammerntheorem)

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

Nach 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.

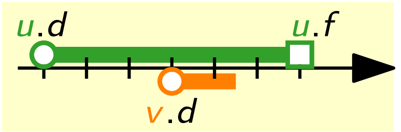
(ii) $[u.d, u.f] \subset [v.d, v.f]$ und Baumkanten enthalten v-u-Weg.

(iii) Wie (ii), nur umgekehrt.

DFSVisit(Graph G, Vertex u)
- $time = time + 1$
- $u.d = time$; $u.color = red$
- **foreach** $v \in \mathrm{Adj}[u]$ **do**
  - **if** $v.color == white$ **then**
    - $v.\pi = u$
    - DFSVisit(G, v)
- $time = time + 1$
- $u.f = time$; $u.color = blue$

Beweis. Wir betrachten zwei Fälle.

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



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

Tiefensuche – Analyse

Satz. (Klammerntheorem)

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

Nach 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.

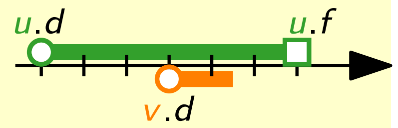
(ii) $[u.d, u.f] \subset [v.d, v.f]$ und Baumkanten enthalten v-u-Weg.

(iii) Wie (ii), nur umgekehrt.

DFSVisit(Graph G, Vertex u)
- $time = time + 1$
- $u.d = time$; $u.color = red$
- **foreach** $v \in \mathrm{Adj}[u]$ **do**
  - **if** $v.color == white$ **then**
    - $v.\pi = u$
    - DFSVisit(G, v)
- $time = time + 1$
- $u.f = time$; $u.color = blue$

Beweis. Wir betrachten zwei Fälle.

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



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

Tiefensuche – Analyse

Satz. (Klammerntheorem)

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

Nach 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.

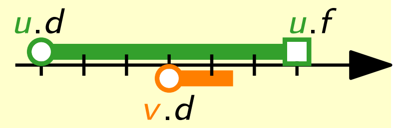
(ii) $[u.d, u.f] \subset [v.d, v.f]$ und Baumkanten enthalten v-u-Weg.

(iii) Wie (ii), nur umgekehrt.

DFSVisit(Graph G, Vertex u)
  $time = time + 1$
  $u.d = time$; $u.color = red$
  foreach $v \in \mathrm{Adj}[u]$ **do**
    if $v.color == white$ **then**
      $v.\pi = u$
      DFSVisit(G, v)
  $time = time + 1$
  $u.f = time$; $u.color = blue$

Beweis. Wir betrachten zwei Fälle.

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



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

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

Tiefensuche – Analyse

Satz. (Klammerntheorem)

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

Nach 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.

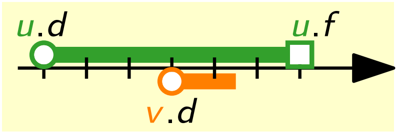
(ii) $[u.d, u.f] \subset [v.d, v.f]$ und Baumkanten enthalten v-u-Weg.

(iii) Wie (ii), nur umgekehrt.

DFSVisit(Graph G, Vertex u)
- $time = time + 1$
- $u.d = time$; $u.color = red$
- **foreach** $v \in \mathrm{Adj}[u]$ **do**
  - **if** $v.color == white$ **then**
    - $v.\pi = u$
    - DFSVisit(G, v)
- $time = time + 1$
- $u.f = time$; $u.color = blue$

Beweis. Wir betrachten zwei Fälle.

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



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

$\Rightarrow$ v ist **Nachfolger** von u , d.h. es gibt einen u-v-Weg.

Tiefensuche – Analyse

Satz. (Klammerntheorem)

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

Nach 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.

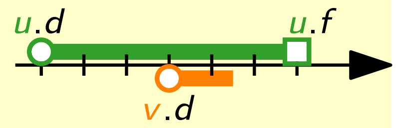
(ii) $[u.d, u.f] \subset [v.d, v.f]$ und Baumkanten enthalten v-u-Weg.

(iii) Wie (ii), nur umgekehrt.

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

Beweis. Wir betrachten zwei Fälle.

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



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

$\Rightarrow$ v ist **Nachfolger** von u , d.h. es gibt einen u-v-Weg.

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

Tiefensuche – Analyse

Satz. (Klammerntheorem)

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

Nach 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.

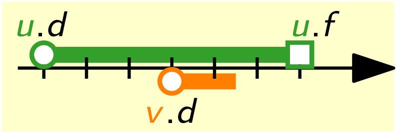
(ii) $[u.d, u.f] \subset [v.d, v.f]$ und Baumkanten enthalten v-u-Weg.

(iii) Wie (ii), nur umgekehrt.

DFSVisit(Graph G, Vertex u)
- $time = time + 1$
- $u.d = time$; $u.color = red$
- **foreach** $v \in \mathrm{Adj}[u]$ **do**
  - **if** $v.color == white$ **then**
    - $v.\pi = u$
    - DFSVisit(G, v)
- $time = time + 1$
- $u.f = time$; $u.color = blue$

Beweis. Wir betrachten zwei Fälle.

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



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

Satz. (Klammerntheorem)

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

Nach 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.

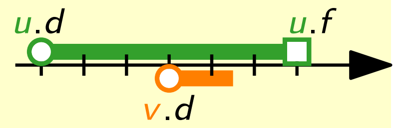
(ii) $[u.d, u.f] \subset [v.d, v.f]$ und Baumkanten enthalten v-u-Weg.

(iii) Wie (ii), nur umgekehrt.

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

Beweis. Wir betrachten zwei Fälle.

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



A) $v.d < u.f$, d.h. v wurde entdeckt, als u noch rot 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 blau, **bevor** DFS zu u zurückkehrt und u blau macht.

Tiefensuche – Analyse

Satz. (Klammerntheorem)

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

Nach 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.

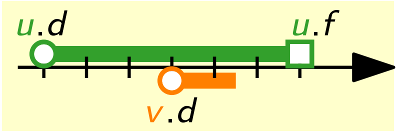
(ii) $[u.d, u.f] \subset [v.d, v.f]$ und Baumkanten enthalten v-u-Weg.

(iii) Wie (ii), nur umgekehrt.

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

Beweis. Wir betrachten zwei Fälle.

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



A) $v.d < u.f$, d.h. v wurde entdeckt, als u noch rot 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 blau, **bevor** DFS zu u zurückkehrt und u blau macht.

$$\Rightarrow [v.d, v.f] \subset [u.d, u.f],$$

Tiefensuche – Analyse

Satz. (Klammerntheorem)

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

Nach 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.

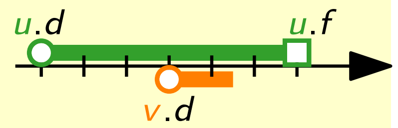
(ii) $[u.d, u.f] \subset [v.d, v.f]$ und Baumkanten enthalten v-u-Weg.

(iii) Wie (ii), nur umgekehrt.

DFSVisit(Graph G, Vertex u)
- $time = time + 1$
- $u.d = time$; $u.color = red$
- **foreach** $v \in \mathrm{Adj}[u]$ **do**
  - **if** $v.color == white$ **then**
    - $v.\pi = u$
    - DFSVisit(G, v)
- $time = time + 1$
- $u.f = time$; $u.color = blue$

Beweis. Wir betrachten zwei Fälle.

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



A) $v.d < u.f$, d.h. v wurde entdeckt, als u noch rot 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 blau, **bevor** DFS zu u zurückkehrt und u blau macht.

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

Tiefensuche – Analyse

Satz. (Klammerntheorem)

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

Nach 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.

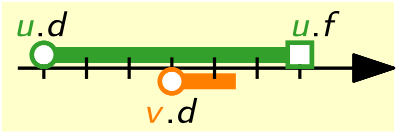
(ii) $[u.d, u.f] \subset [v.d, v.f]$ und Baumkanten enthalten v-u-Weg.

(iii) Wie (ii), nur umgekehrt.

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

Beweis. Wir betrachten zwei Fälle.

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



A) $v.d < u.f$. ✓

Tiefensuche – Analyse

Satz. (Klammerntheorem)

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

Nach 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.

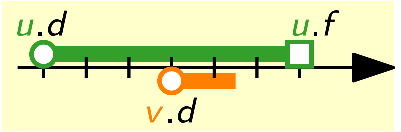
(ii) $[u.d, u.f] \subset [v.d, v.f]$ und Baumkanten enthalten v-u-Weg.

(iii) Wie (ii), nur umgekehrt.

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

Beweis. Wir betrachten zwei Fälle.

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



A) $v.d < u.f$. ✓

B)

Tiefensuche – Analyse

Satz. (Klammerntheorem)

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

Nach 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.

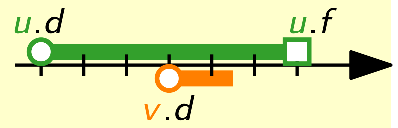
(ii) $[u.d, u.f] \subset [v.d, v.f]$ und Baumkanten enthalten v-u-Weg.

(iii) Wie (ii), nur umgekehrt.

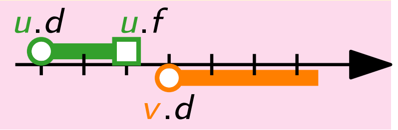
DFSVisit(Graph G, Vertex u)
$\quad time = time + 1$
$\quad u.d = time$; $u.color = red$
$\quad$**foreach** $v \in \mathrm{Adj}[u]$ **do**
$\quad\quad$**if** $v.color == white$ **then**
$\quad\quad\quad v.\pi = u$
$\quad\quad\quad$DFSVisit(G, v)
$\quad time = time + 1$
$\quad u.f = time$; $u.color = blue$

Beweis. Wir betrachten zwei Fälle.

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



A) $v.d < u.f$. ✓



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

Tiefensuche – Analyse

Satz. (Klammerntheorem)

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

Nach 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.

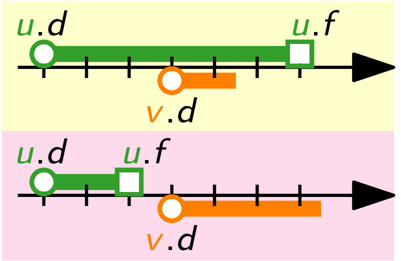
(ii) $[u.d, u.f] \subset [v.d, v.f]$ und Baumkanten enthalten v-u-Weg.

(iii) Wie (ii), nur umgekehrt.

DFSVisit(Graph G, Vertex u)
- $time = time + 1$
- $u.d = time$; $u.color = red$
- **foreach** $v \in \mathrm{Adj}[u]$ **do**
  - **if** $v.color == white$ **then**
    - $v.\pi = u$
    - DFSVisit(G, v)
- $time = time + 1$
- $u.f = time$; $u.color = blue$

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

Satz. (Klammerntheorem)

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

Nach 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.

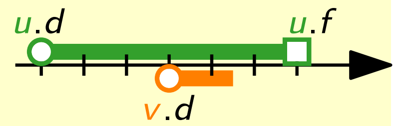
(ii) $[u.d, u.f] \subset [v.d, v.f]$ und Baumkanten enthalten v-u-Weg.

(iii) Wie (ii), nur umgekehrt.

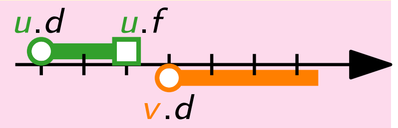
```
DFSVisit(Graph G, Vertex u)
  time = time + 1
  u.d = time; u.color = red
  foreach v ∈ Adj[u] do
    if v.color == white then
      v.π = u
      DFSVisit(G, v)
  time = time + 1
  u.f = time; u.color = blue
```

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

Satz. (Klammerntheorem)

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

Nach 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.

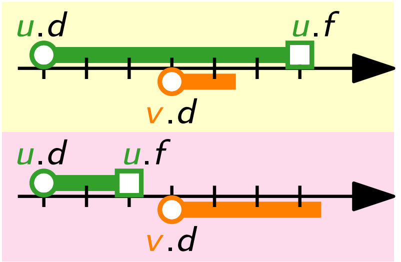
(ii) $[u.d, u.f] \subset [v.d, v.f]$ und Baumkanten enthalten v-u-Weg.

(iii) Wie (ii), nur umgekehrt.

DFSVisit(Graph G, Vertex u)
  $time = time + 1$
  $u.d = time$; $u.color = red$
  foreach $v \in \text{Adj}[u]$ **do**
    if $v.color == white$ **then**
      $v.\pi = u$
      DFSVisit(G, v)
  $time = time + 1$
  $u.f = time$; $u.color = blue$

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

Satz. (Klammerntheorem)

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

Nach 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.

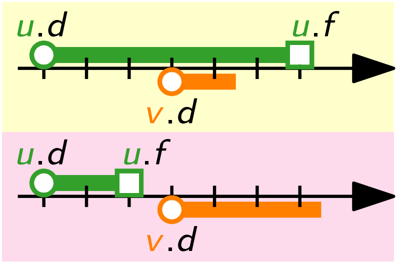
(ii) $[u.d, u.f] \subset [v.d, v.f]$ und Baumkanten enthalten v-u-Weg.

(iii) Wie (ii), nur umgekehrt.

DFSVisit(Graph G, Vertex u)
  $time = time + 1$
  $u.d = time$; $u.color = red$
  foreach $v \in \mathrm{Adj}[u]$ **do**
    if $v.color == white$ **then**
      $v.\pi = u$
      DFSVisit(G, v)
  $time = time + 1$
  $u.f = time$; $u.color = blue$

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

Satz. (Klammerntheorem)

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

Nach 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.

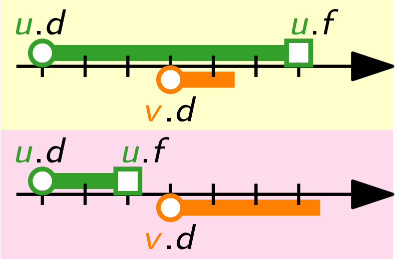
(ii) $[u.d, u.f] \subset [v.d, v.f]$ und Baumkanten enthalten v-u-Weg.

(iii) Wie (ii), nur umgekehrt.

DFSVisit(Graph G, Vertex u)
  $time = time + 1$
  $u.d = time$; $u.color = red$
  foreach $v \in \mathrm{Adj}[u]$ **do**
    if $v.color == white$ **then**
      $v.\pi = u$
      DFSVisit(G, v)
  $time = time + 1$
  $u.f = time$; $u.color = blue$

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 $u.d < u.f < v.d < v.f$

$\Rightarrow$

Tiefensuche – Analyse

Satz. (Klammerntheorem)

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

Nach 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.

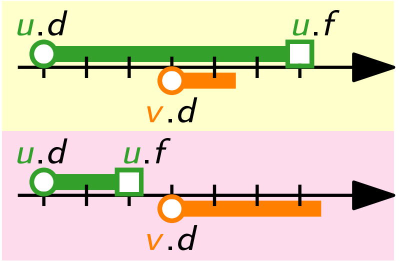
(iii) Wie (ii), nur umgekehrt.

DFSVisit(Graph G, Vertex u)

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

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

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

Satz. (Klammerntheorem)
Nach 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.

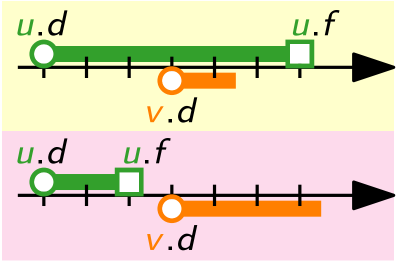
(ii) $[u.d, u.f] \subset [v.d, v.f]$ und Baumkanten enthalten v-u-Weg.

(iii) Wie (ii), nur umgekehrt.

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

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 rot war.

Tiefensuche – Analyse

Satz. (Klammerntheorem)

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

Nach 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.

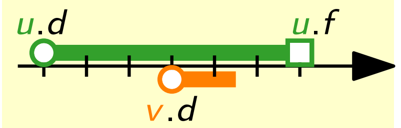
(ii) $[u.d, u.f] \subset [v.d, v.f]$ und Baumkanten enthalten v-u-Weg.

(iii) Wie (ii), nur umgekehrt.

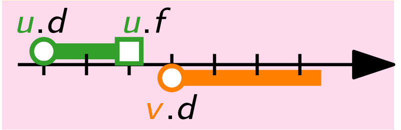
```
DFSVisit(Graph G, Vertex u)
  time = time + 1
  u.d = time; u.color = red
  foreach v ∈ Adj[u] do
    if v.color == white then
      v.π = u
      DFSVisit(G, v)
  time = time + 1
  u.f = time; u.color = blue
```

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 rot war, d.h. keiner ist Nachfolger des anderen.

Tiefensuche – Analyse

Satz. (Klammerntheorem)

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

Nach 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.

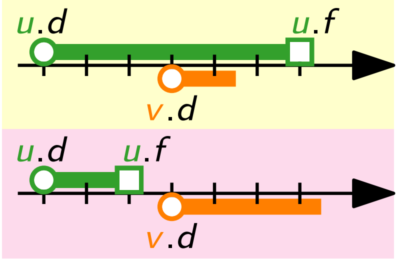
(ii) $[u.d, u.f] \subset [v.d, v.f]$ und Baumkanten enthalten v-u-Weg.

(iii) Wie (ii), nur umgekehrt.

DFSVisit(Graph G, Vertex u)
  $time = time + 1$
  $u.d = time$; $u.color = red$
  foreach $v \in \mathrm{Adj}[u]$ **do**
    if $v.color == white$ **then**
      $v.\pi = u$
      DFSVisit(G, v)
  $time = time + 1$
  $u.f = time$; $u.color = blue$

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 rot war, d.h. keiner ist Nachfolger des anderen. $\Rightarrow$ (i)

Tiefensuche – Analyse

Satz. (Klammerntheorem)

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

Nach 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.

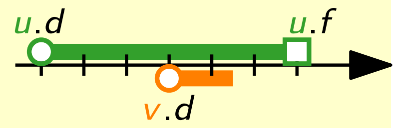
(ii) $[u.d, u.f] \subset [v.d, v.f]$ und Baumkanten enthalten v-u-Weg.

(iii) Wie (ii), nur umgekehrt.

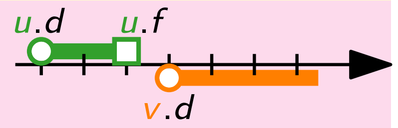
```
DFSVisit(Graph G, Vertex u)
  time = time + 1
  u.d = time; u.color = red
  foreach v ∈ Adj[u] do
    if v.color == white then
      v.π = u
      DFSVisit(G, v)
  time = time + 1
  u.f = time; u.color = blue
```

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 rot war, d.h. keiner ist Nachfolger des anderen. $\Rightarrow$ (i)

Tiefensuche – Analyse

Satz. (Klammerntheorem)

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

Nach 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.

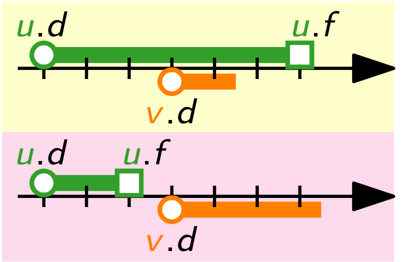
(ii) $[u.d, u.f] \subset [v.d, v.f]$ und Baumkanten enthalten v-u-Weg.

(iii) Wie (ii), nur umgekehrt.

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

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 rot war, d.h. keiner ist Nachfolger des anderen. $\Rightarrow$ (i)

Tiefensuche – Analyse

Satz. (Klammerntheorem)

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

Nach 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.

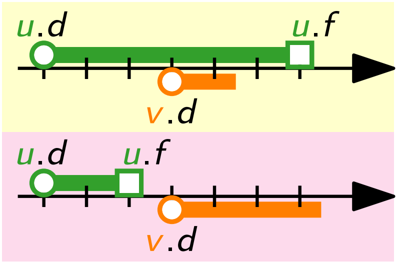
(ii) $[u.d, u.f] \subset [v.d, v.f]$ und Baumkanten enthalten v-u-Weg.

(iii) Wie (ii), nur umgekehrt.

DFSVisit(Graph G, Vertex u)
- $time = time + 1$
- $u.d = time$; $u.color = red$
- **foreach** $v \in \mathrm{Adj}[u]$ **do**
  - **if** $v.color == white$ **then**
    - $v.\pi = u$
    - DFSVisit(G, v)
- $time = time + 1$
- $u.f = time$; $u.color = blue$

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 rot war, d.h. keiner ist Nachfolger des anderen. $\Rightarrow$ (i)

Tiefensuche – Analyse

Satz. (Klammerntheorem)

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

Nach 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.

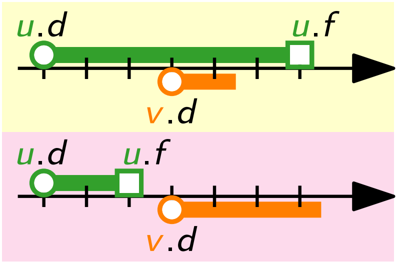
(ii) $[u.d, u.f] \subset [v.d, v.f]$ und Baumkanten enthalten v-u-Weg.

(iii) Wie (ii), nur umgekehrt.

DFSVisit(Graph G, Vertex u)
- $time = time + 1$
- $u.d = time$; $u.color = red$
- **foreach** $v \in \mathrm{Adj}[u]$ **do**
  - **if** $v.color == white$ **then**
    - $v.\pi = u$
    - DFSVisit(G, v)
- $time = time + 1$
- $u.f = time$; $u.color = blue$

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 rot war, d.h. keiner ist Nachfolger des anderen. $\Rightarrow$ (i)

Tiefensuche – Analyse

Satz. (Klammerntheorem)

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

Nach 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.

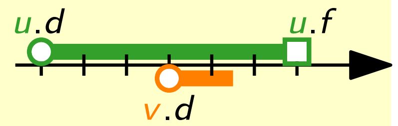
(iii) Wie (ii), nur umgekehrt.

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

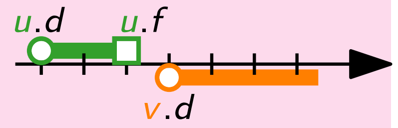
Beweis. Wir betrachten zwei Fälle.

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 rot war, d.h. keiner ist Nachfolger des anderen. $\Rightarrow$ (i)

Tiefensuche – Analyse

Satz. (Klammerntheorem)

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

Nach 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.

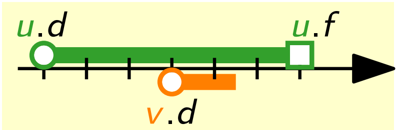
(iii) Wie (ii), nur umgekehrt.

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

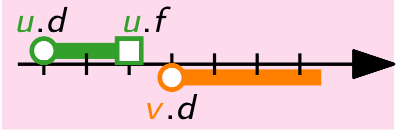
Beweis. Wir betrachten zwei Fälle.

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 rot war, d.h. keiner ist Nachfolger des anderen. $\Rightarrow$ (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 blau,
bevor u blau 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 blau,
bevor u blau gefärbt wird (da $v \in \mathrm{Adj}[u]$).

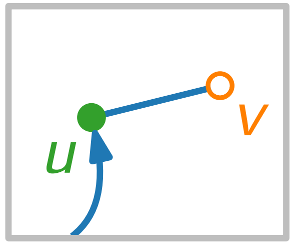
Tiefensuche in ungerichteten Graphen

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 blau,
bevor u blau gefärbt wird (da $v \in \mathrm{Adj}[u]$).



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

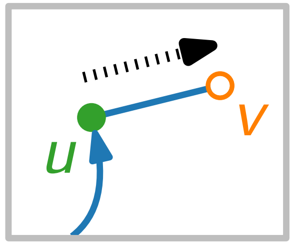
Tiefensuche in ungerichteten Graphen

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 blau,
bevor u blau gefärbt wird (da $v \in \mathrm{Adj}[u]$).



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

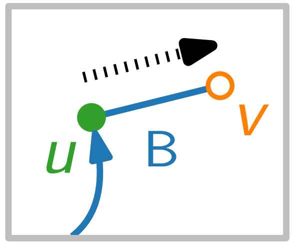
Tiefensuche in ungerichteten Graphen

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 blau,
bevor u blau gefärbt wird (da $v \in \mathrm{Adj}[u]$).



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

  Dann ist uv Baumkante.

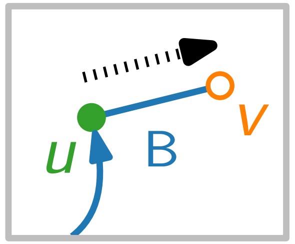
Tiefensuche in ungerichteten Graphen

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

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

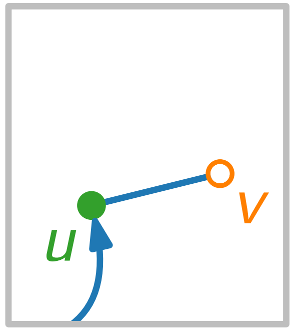
O.B.d.A. gilt $u.d < v.d$.

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



- 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

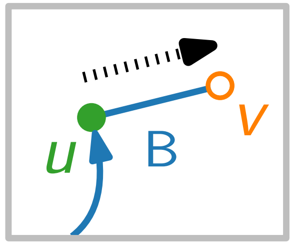
Tiefensuche in ungerichteten Graphen

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

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

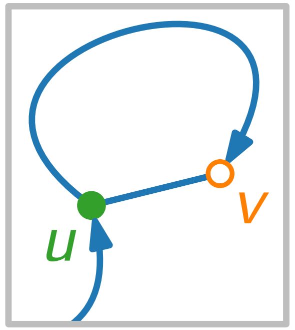
O.B.d.A. gilt $u.d < v.d$.

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



- 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

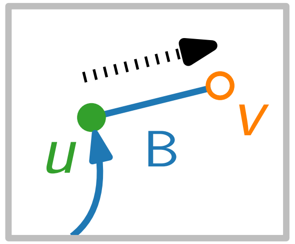
Tiefensuche in ungerichteten Graphen

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

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

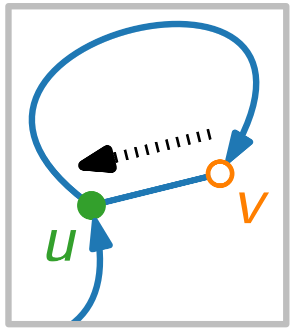
O.B.d.A. gilt $u.d < v.d$.

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



- 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

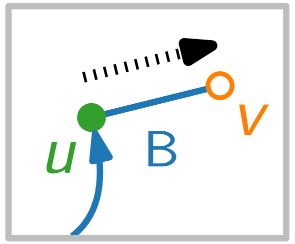
Tiefensuche in ungerichteten Graphen

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

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

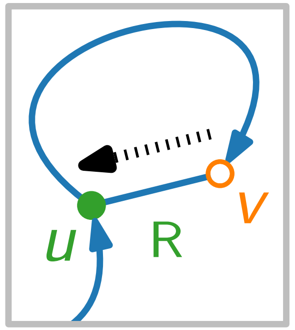
O.B.d.A. gilt $u.d < v.d$.

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



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

  Dann ist uv Baumkante.



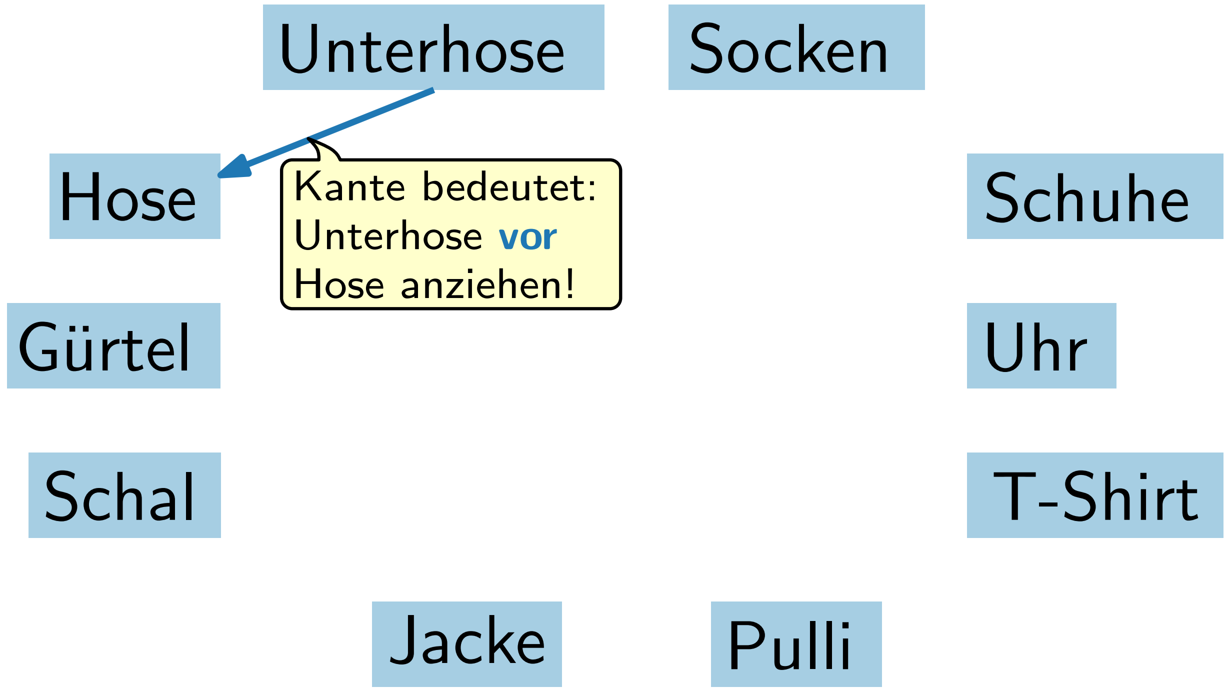
- Andernfalls wird uv zum ersten Mal von v nach u überschritten. Dann ist uv R-Kante, da u dann schon (und immer noch) rot ist.

$\square$

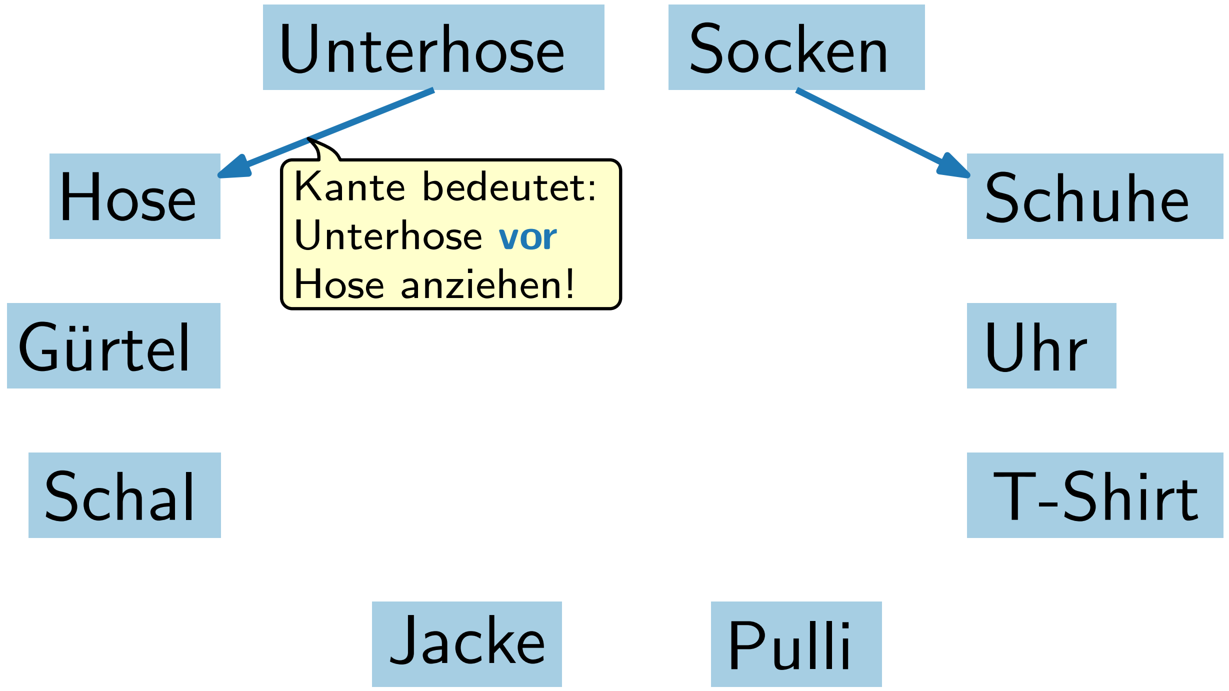
Ablaufplanung

Unterhose
Socken
Hose
Schuhe
Gürtel
Uhr
Schal
T-Shirt
Jacke
Pulli

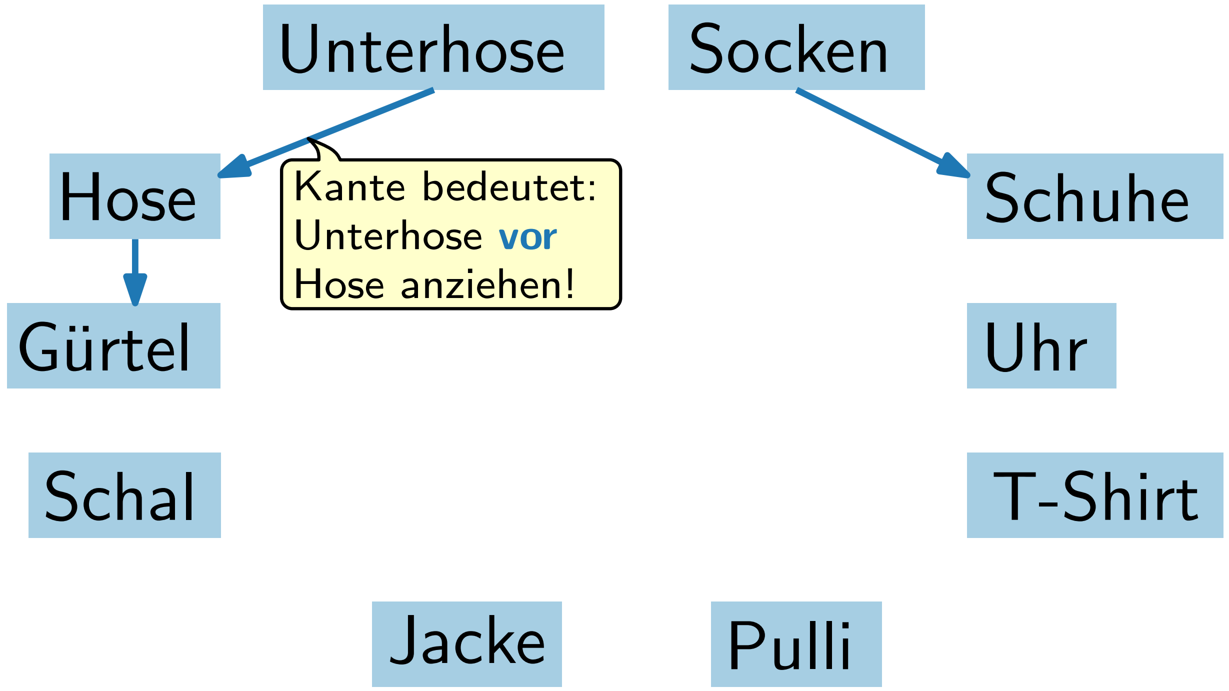
Ablaufplanung



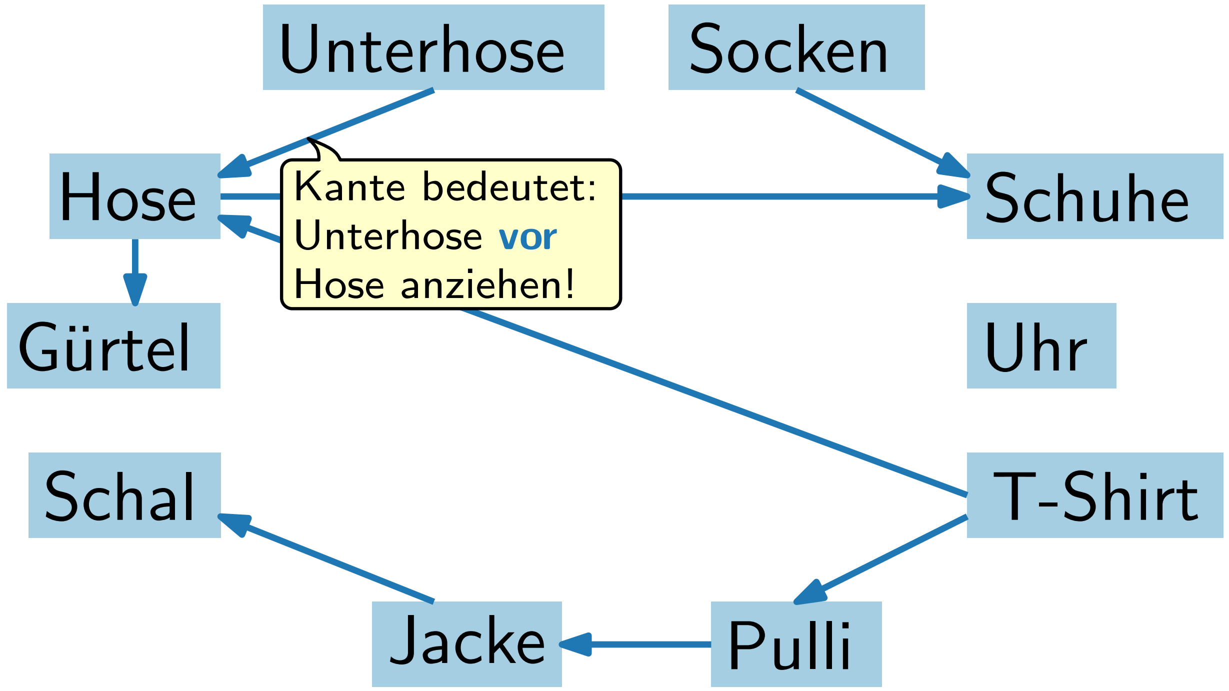
Ablaufplanung



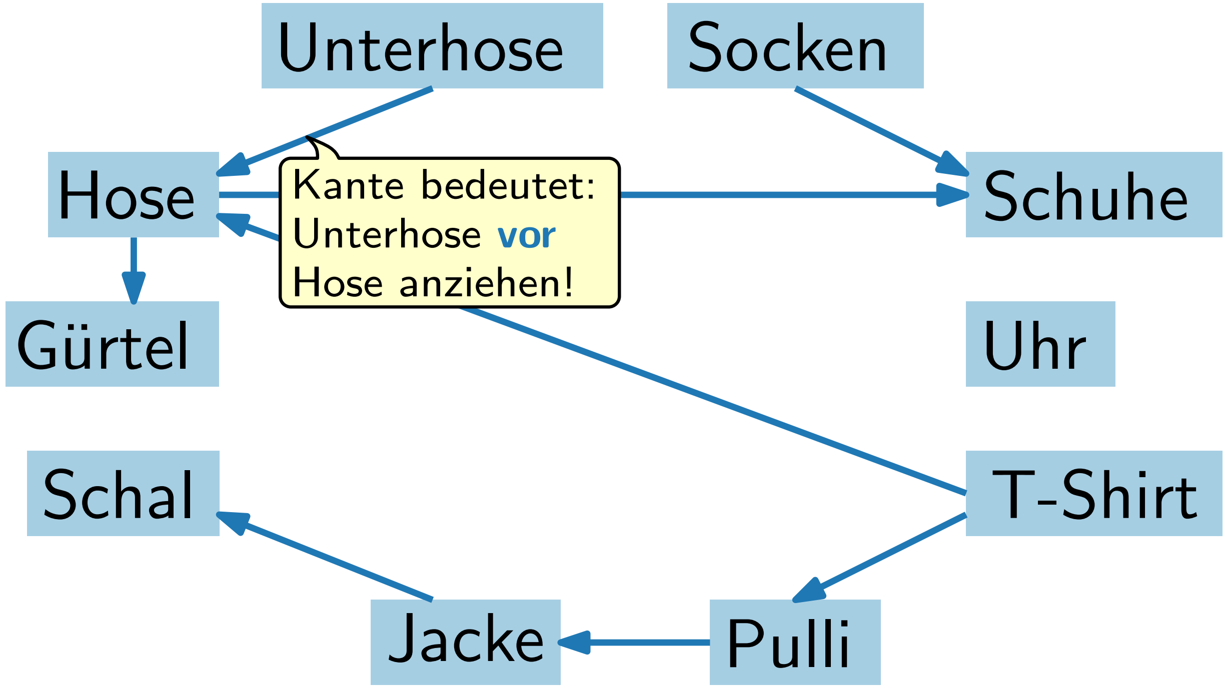
Ablaufplanung



Ablaufplanung



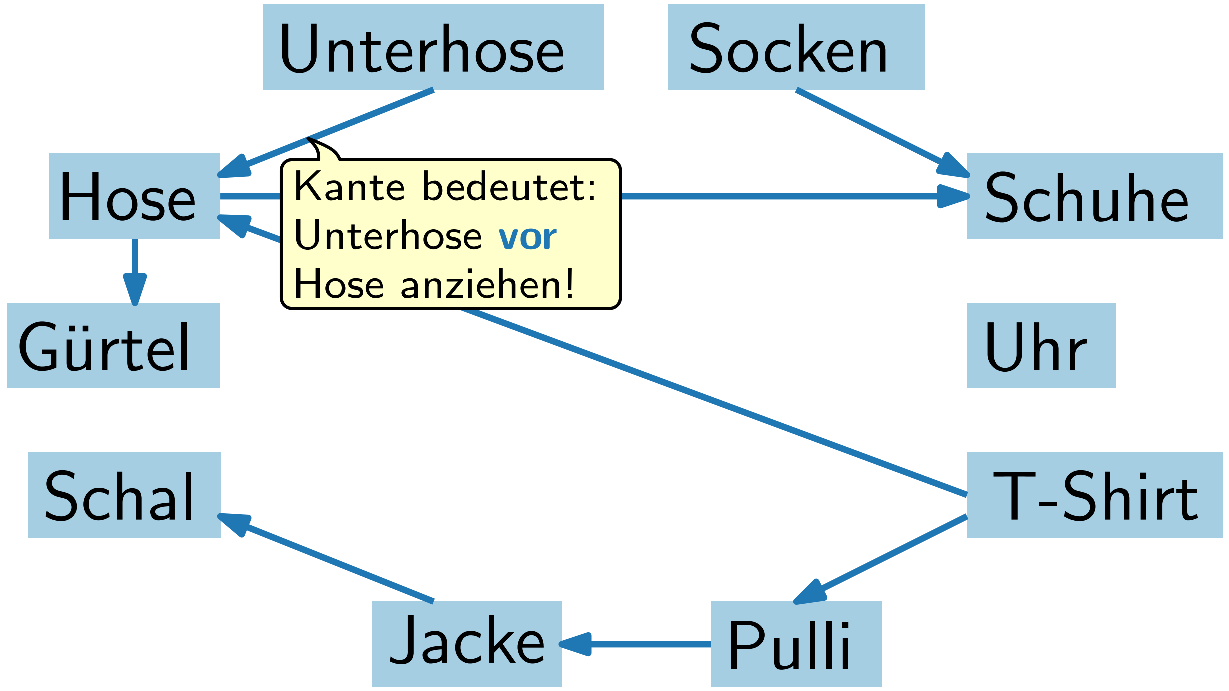
Ablaufplanung



Aufgabe: Finde Ablaufplan –

d.h. Reihenfolge der Knoten, so dass alle Einschränkungen erfüllt sind (z.B. T-Shirt vor Pulli).

Ablaufplanung

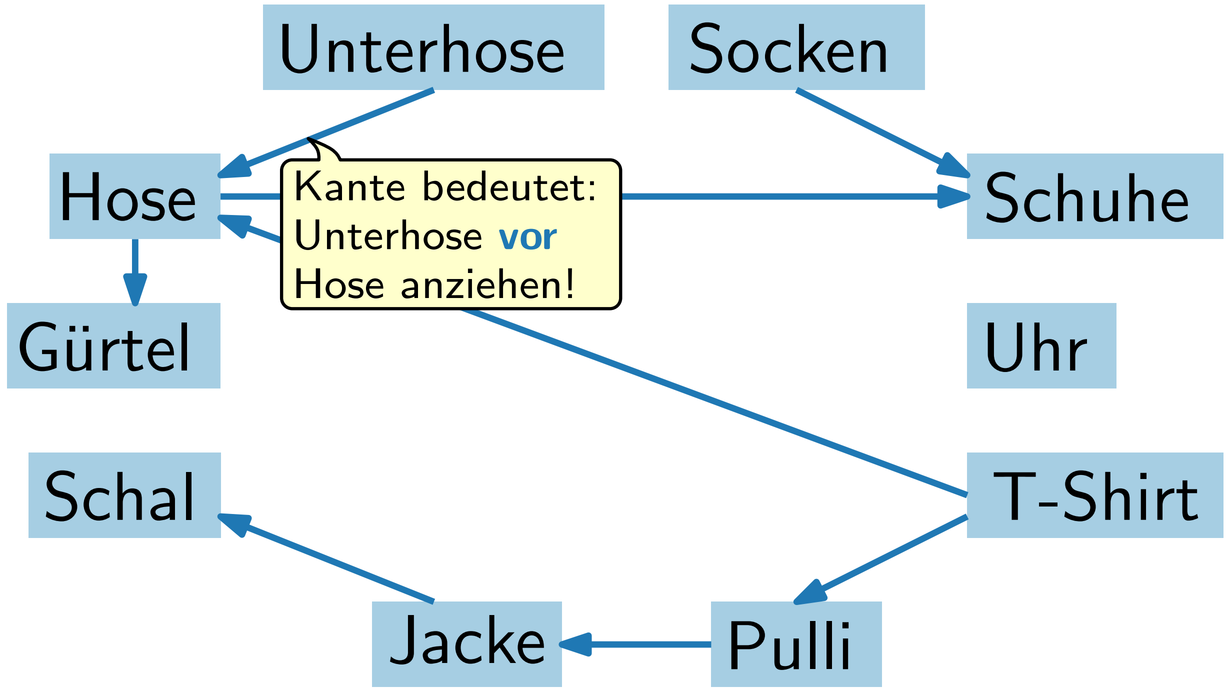


Aufgabe: Finde Ablaufplan –

d.h. Reihenfolge der Knoten, so dass alle Einschränkungen erfüllt sind (z.B. T-Shirt vor Pulli).

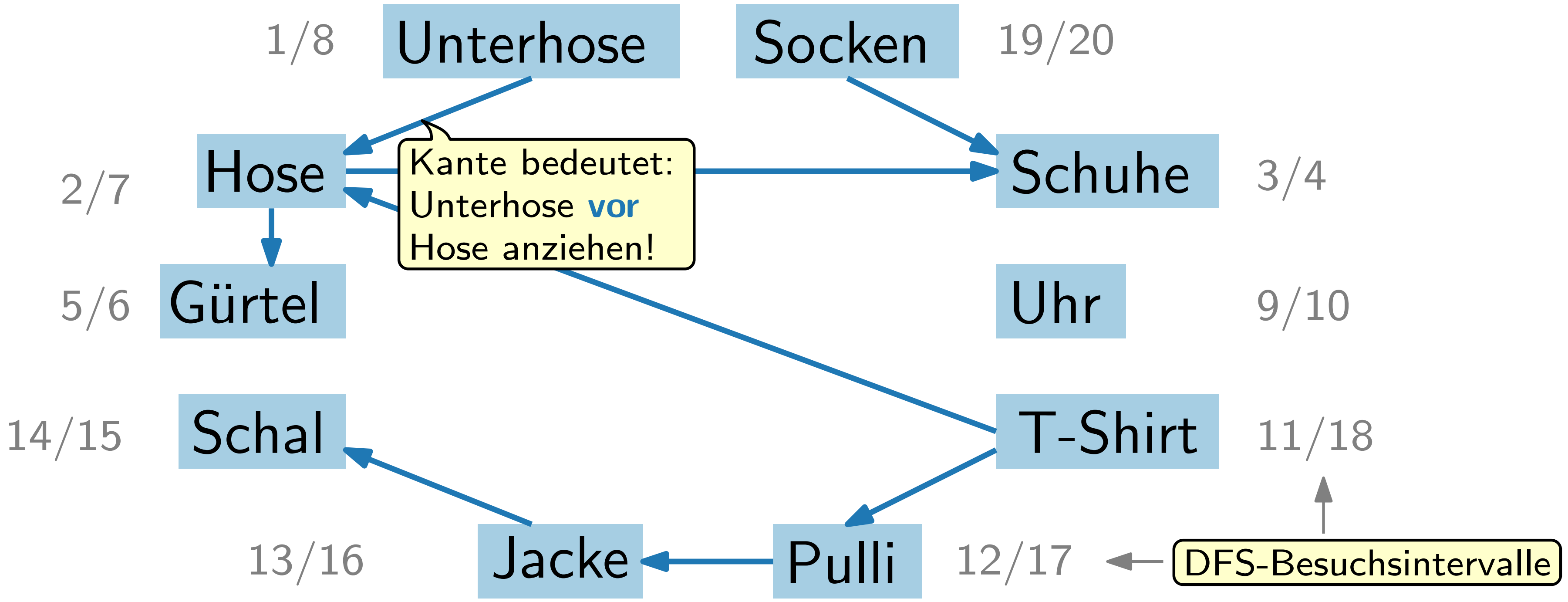
Topologische Sortierung: Lineare Ordnung der Knoten, so dass aus $(u, v) \in E(G)$ folgt: u kommt vor v.

Ablaufplanung



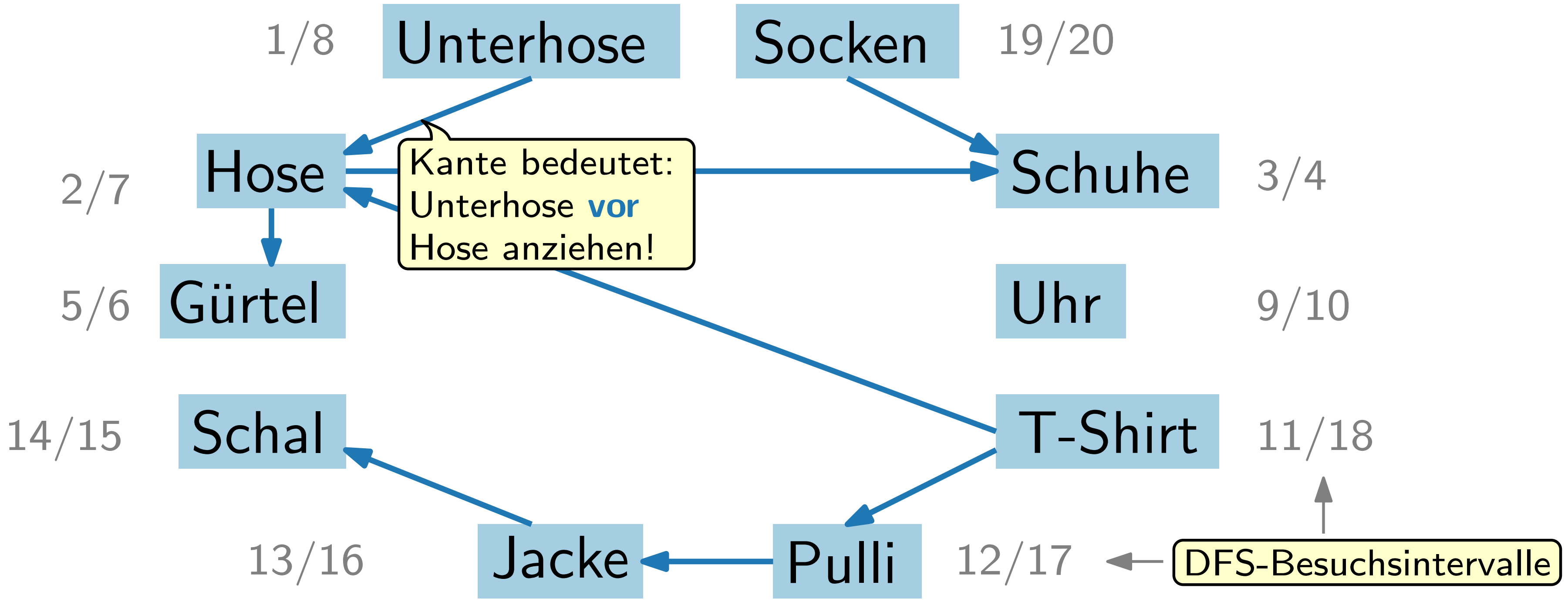
Idee: Nutze Tiefensuche!

Ablaufplanung



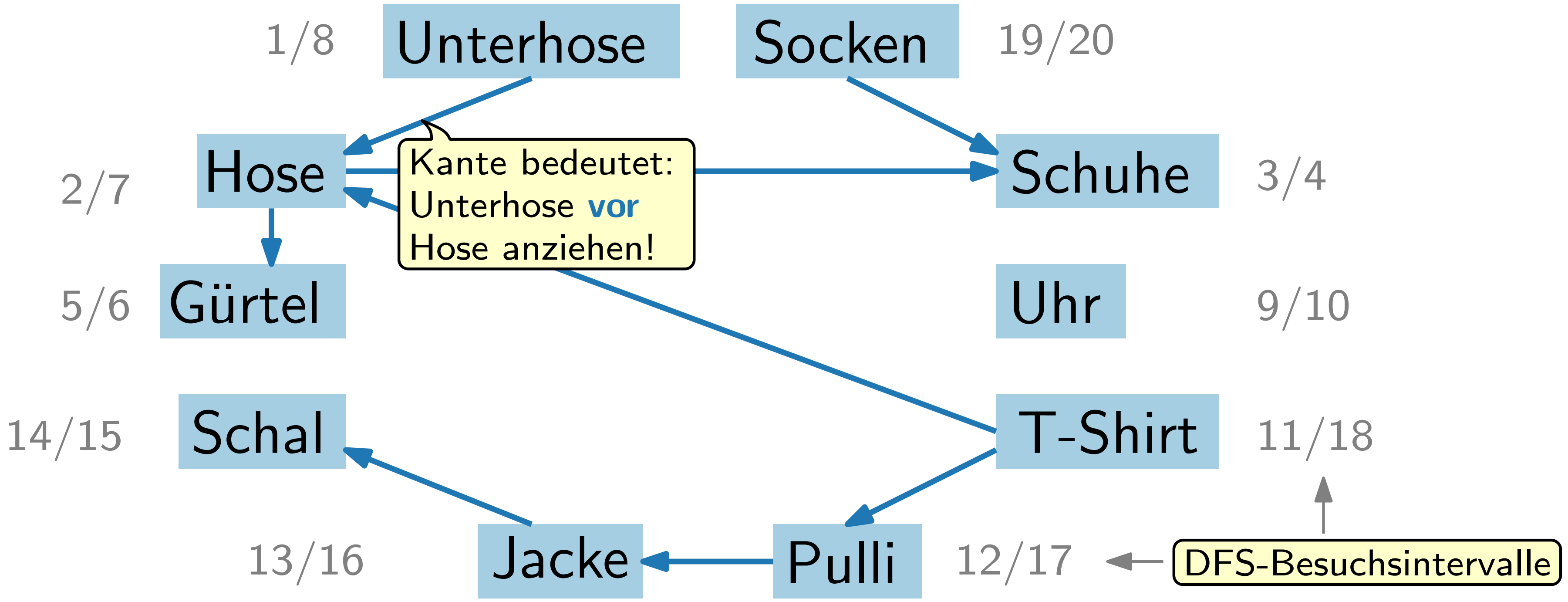
Idee: Nutze Tiefensuche!

Ablaufplanung



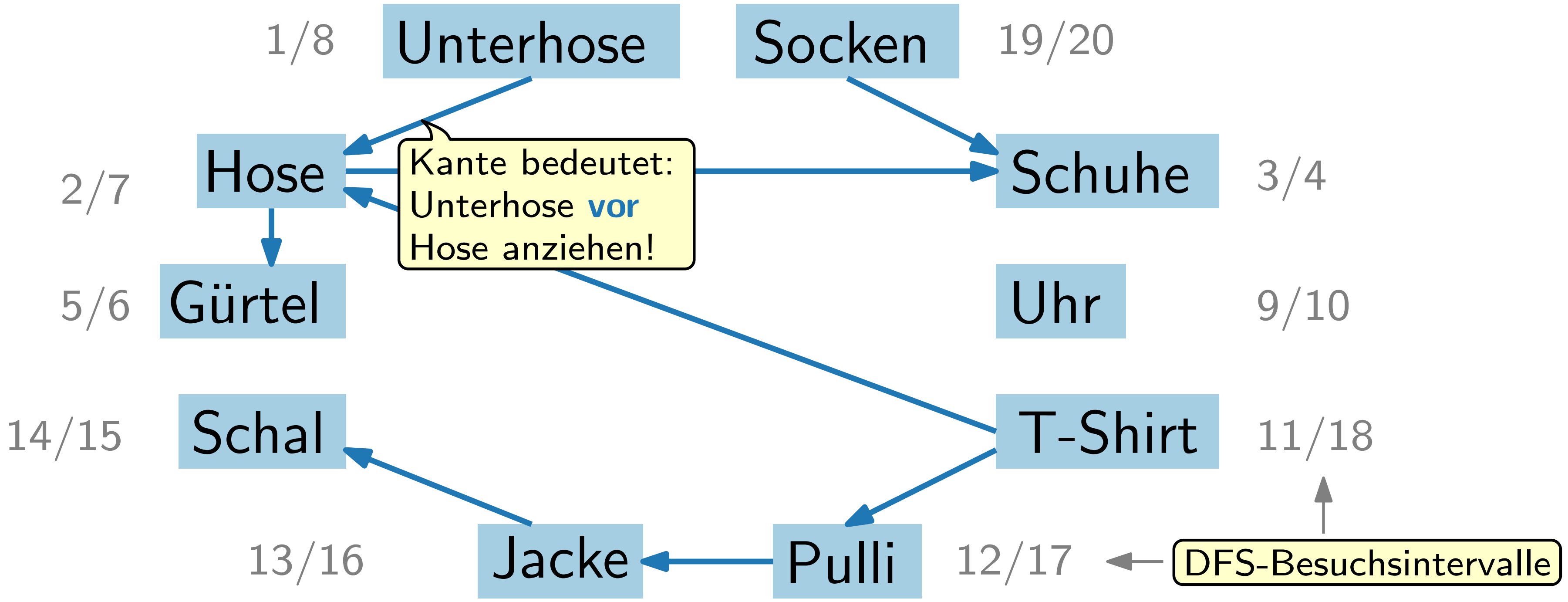
Idee: Nutze Tiefensuche!
Sortiere Knoten nach

Ablaufplanung



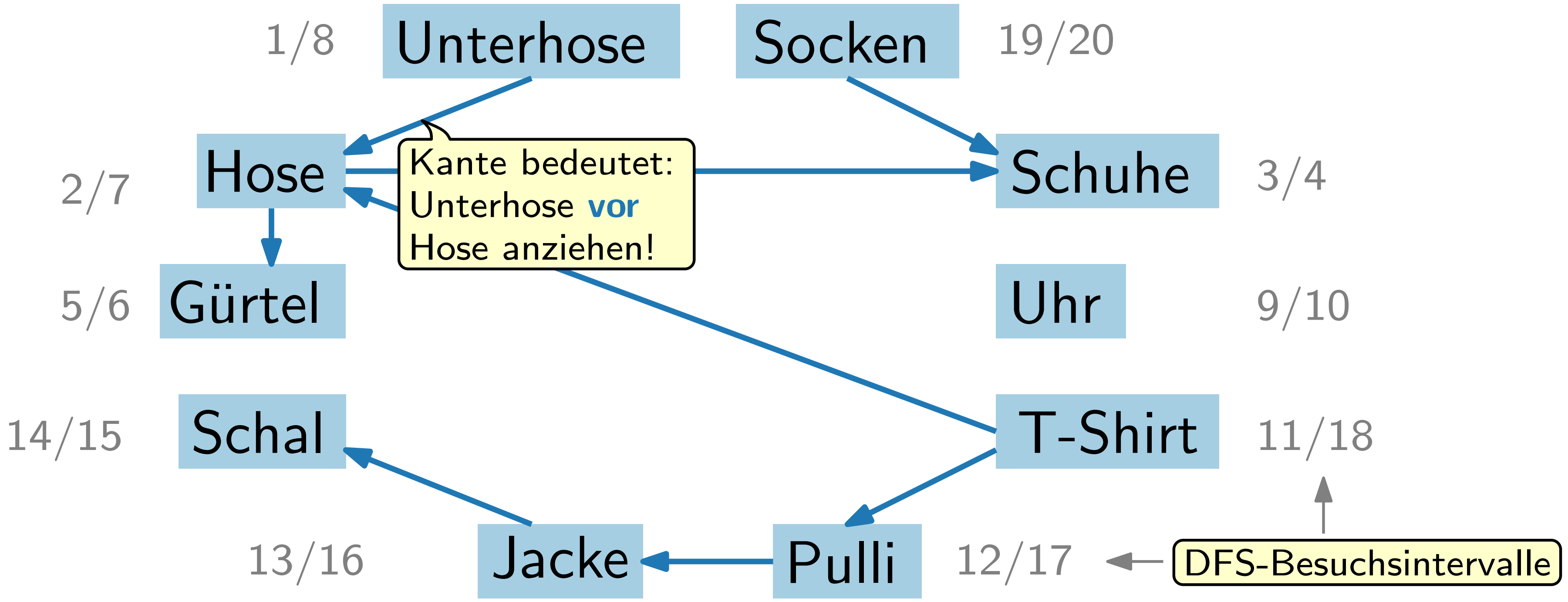
Idee: Nutze Tiefensuche!
Sortiere Knoten nach absteigenden f-Zeiten.

Ablaufplanung

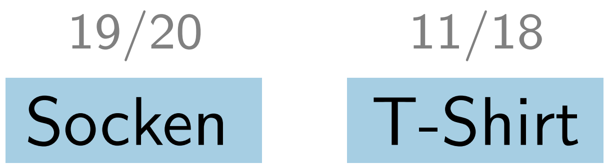


Idee: Nutze Tiefensuche!
Sortiere Knoten nach absteigenden f-Zeiten.

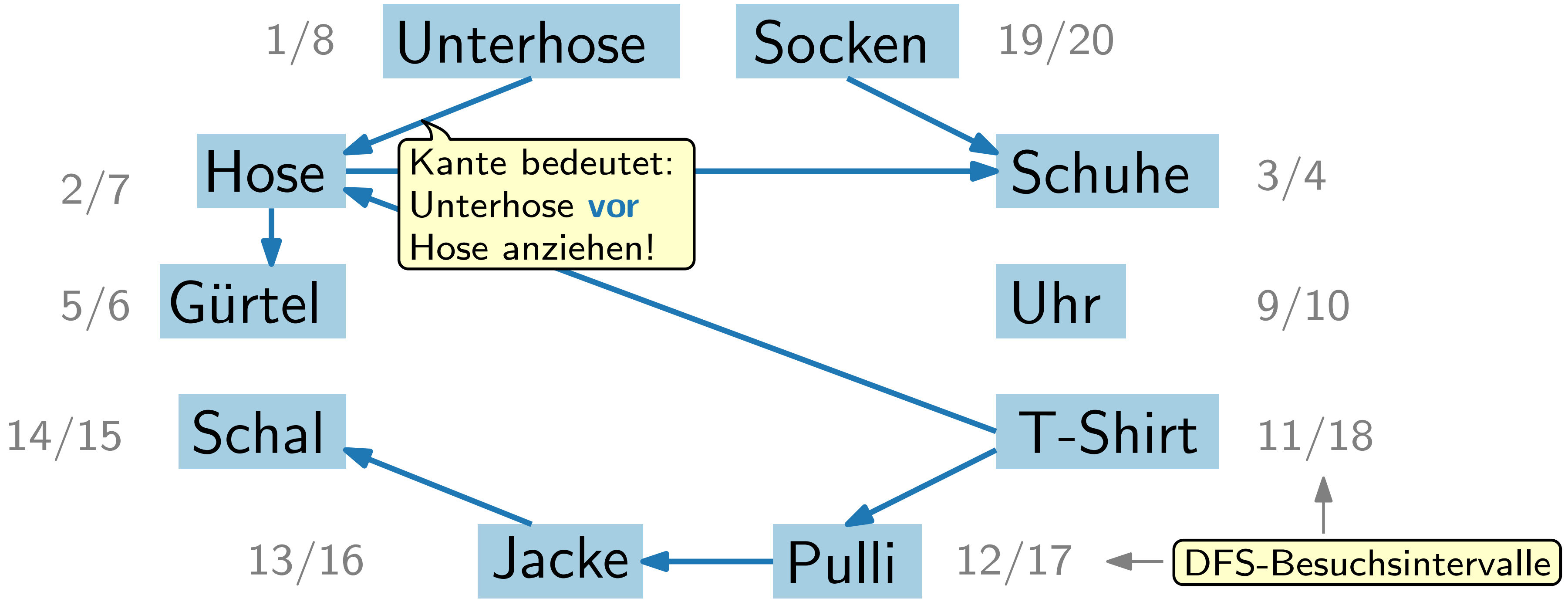
Ablaufplanung



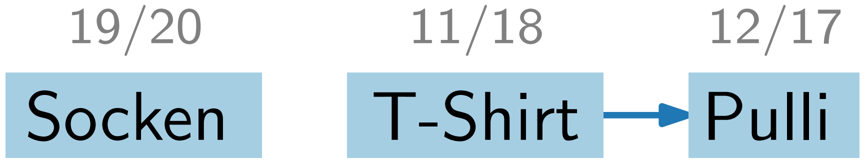
Idee: Nutze Tiefensuche!
Sortiere Knoten nach absteigenden f-Zeiten.



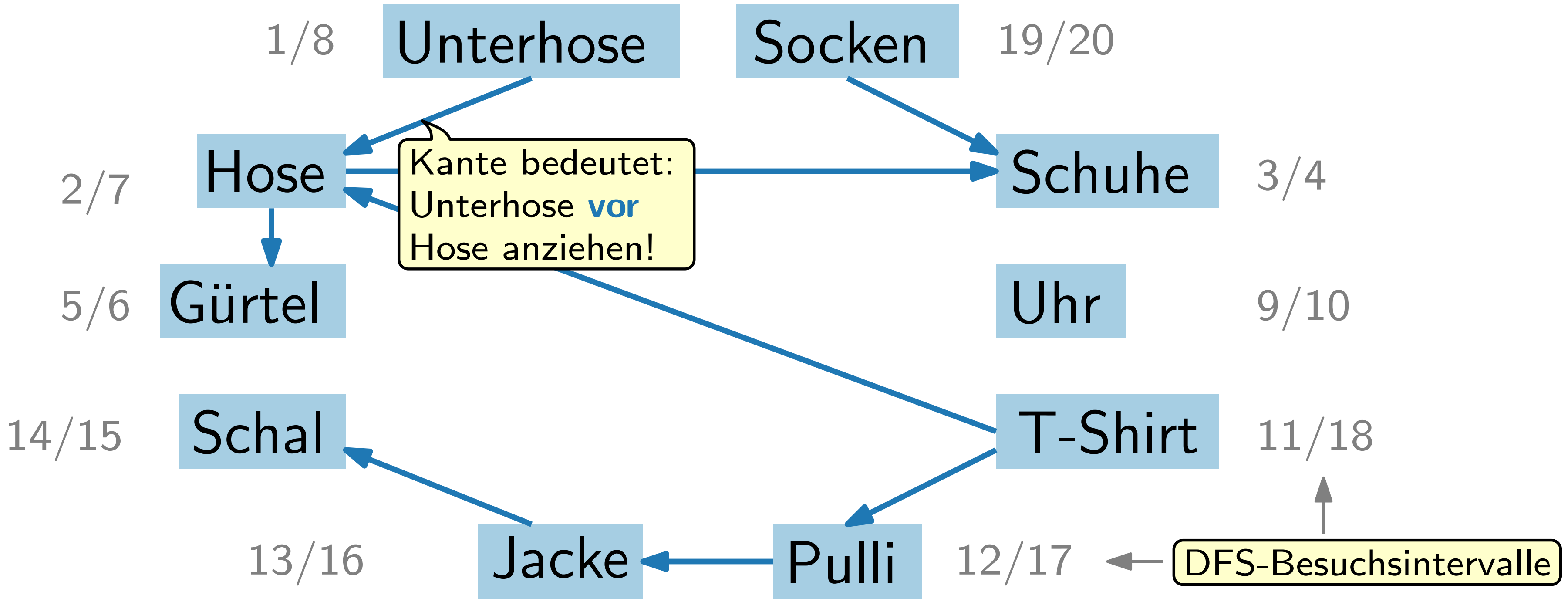
Ablaufplanung



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

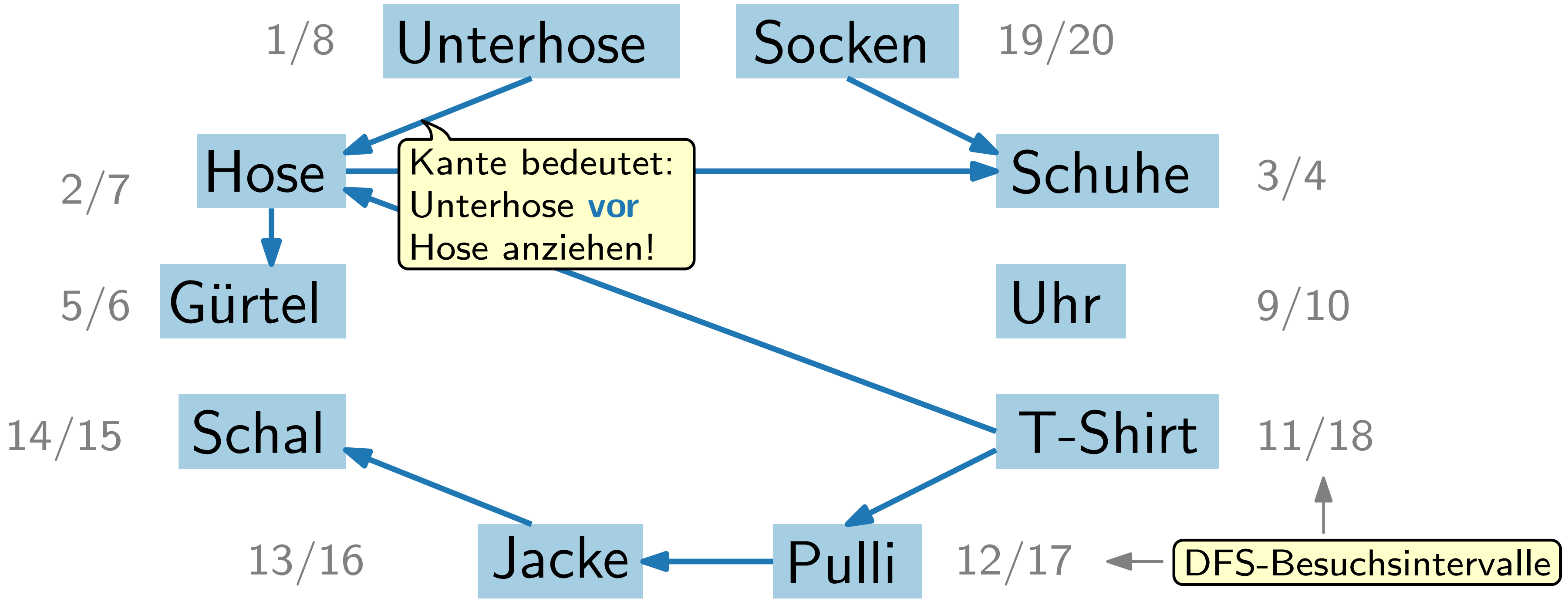


Ablaufplanung

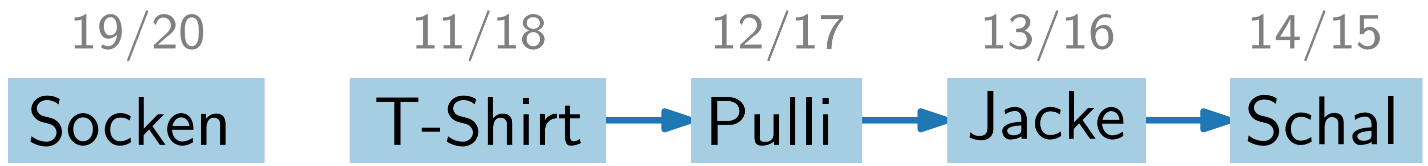


Idee: Nutze Tiefensuche!
Sortiere Knoten nach absteigenden f-Zeiten.

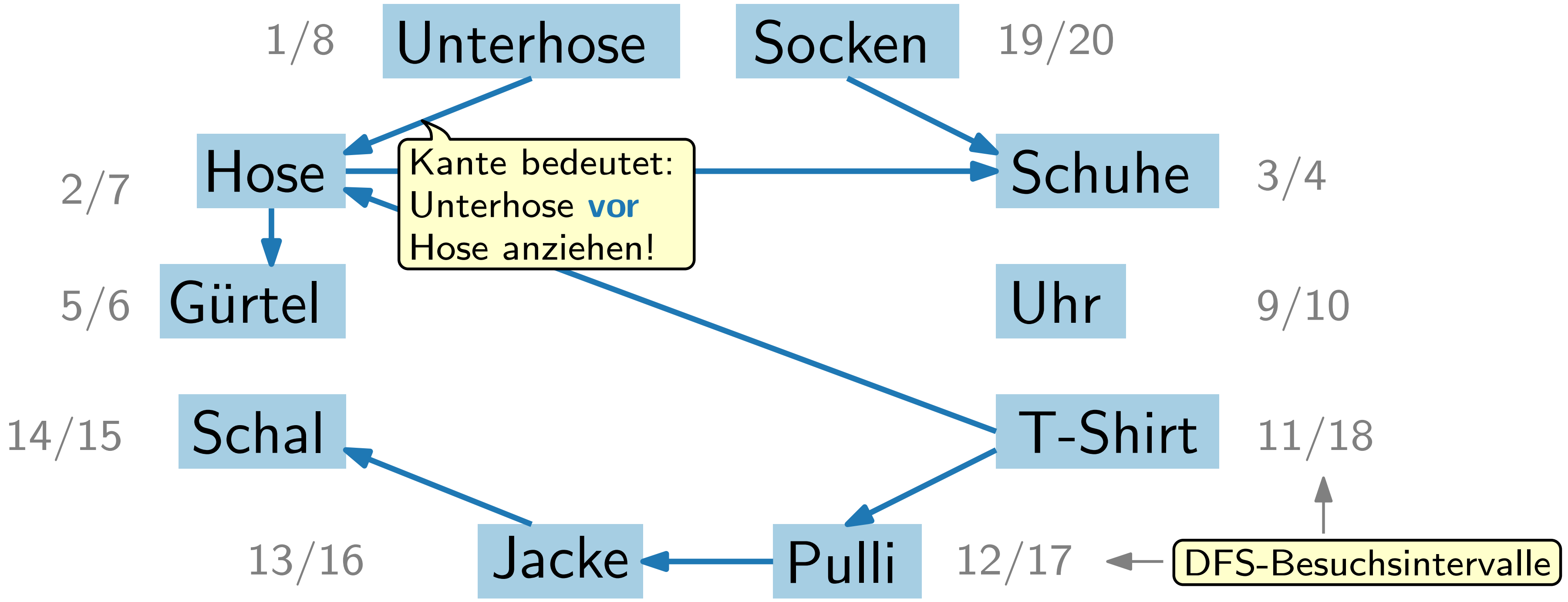
Ablaufplanung



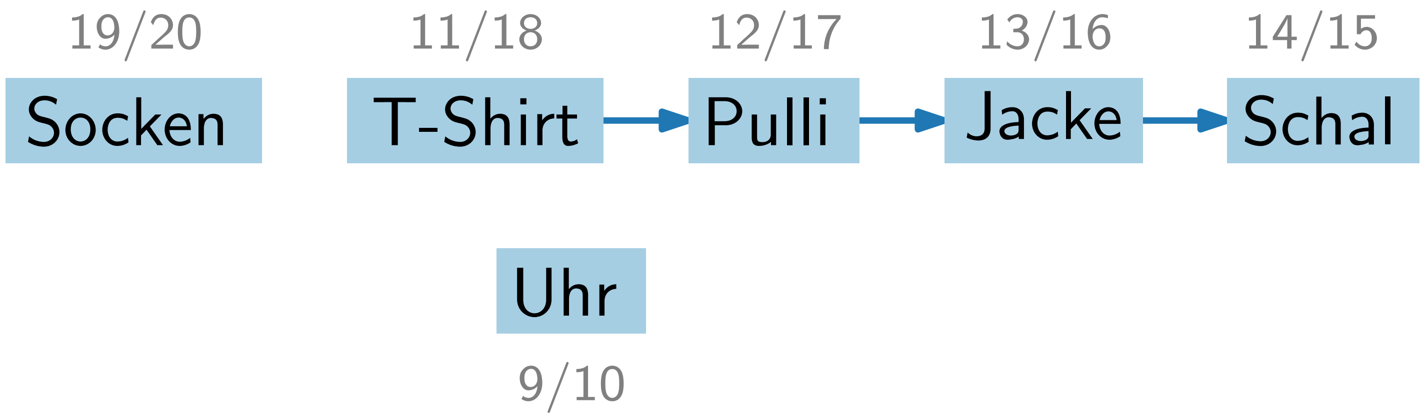
Idee: Nutze Tiefensuche!
Sortiere Knoten nach absteigenden f-Zeiten.



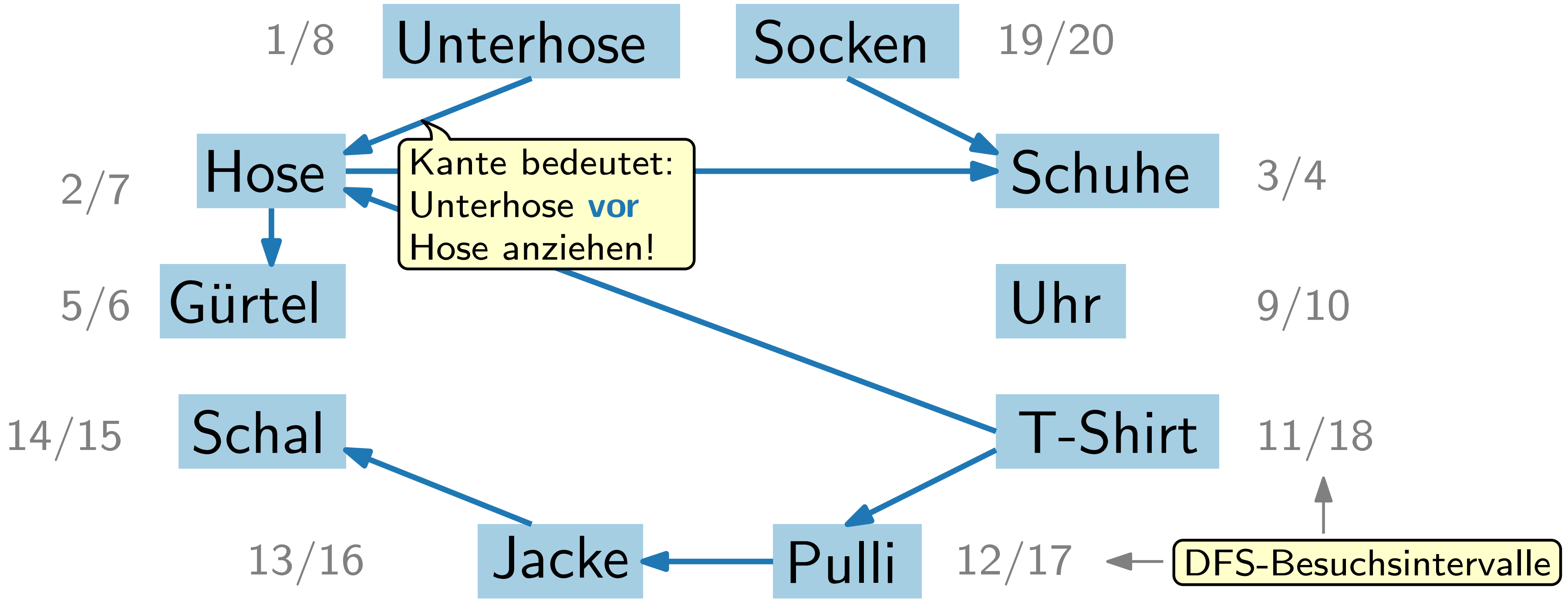
Ablaufplanung



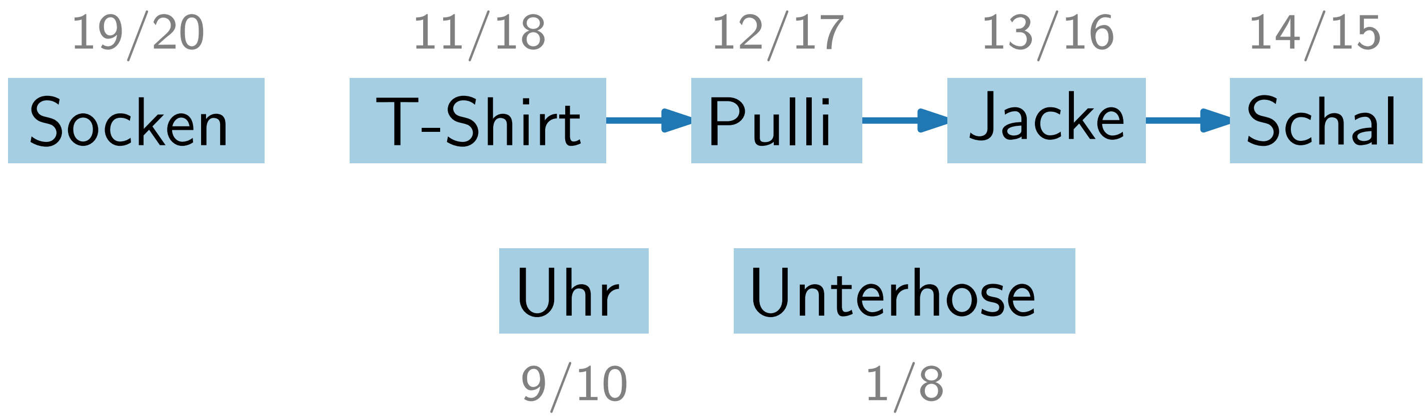
Idee: Nutze Tiefensuche!
Sortiere Knoten nach absteigenden f-Zeiten.



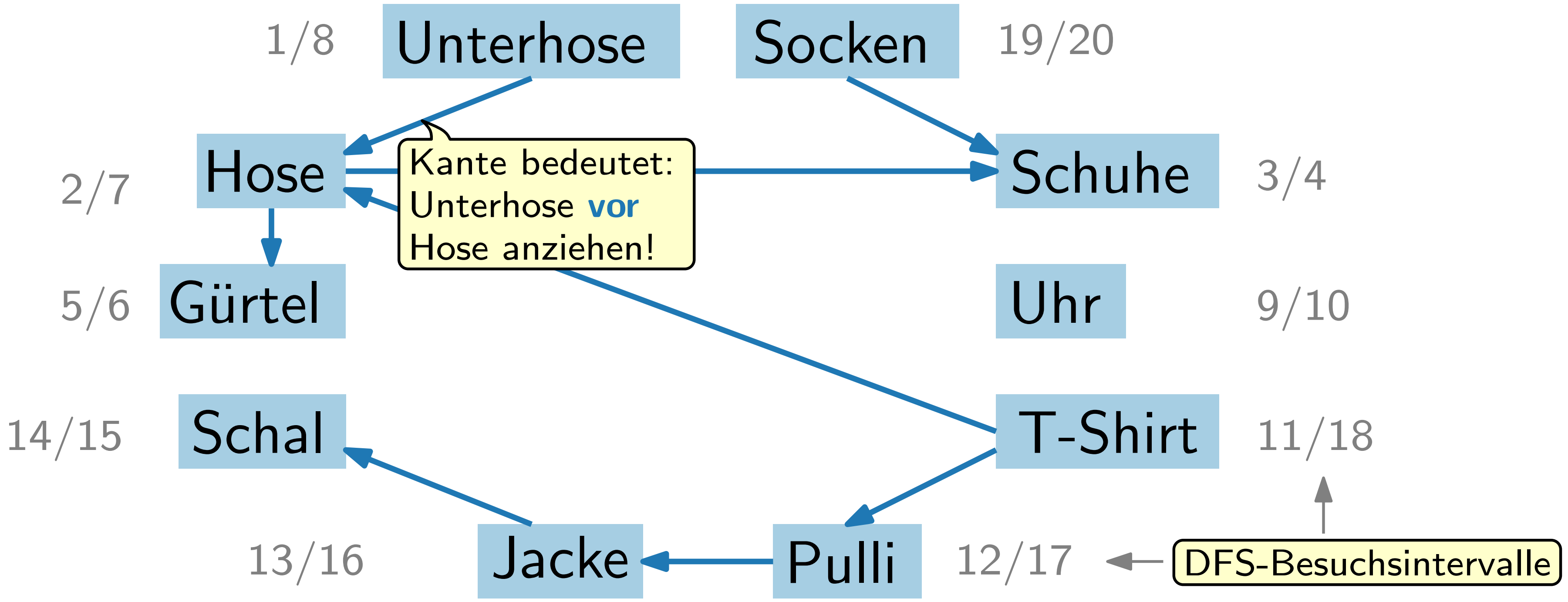
Ablaufplanung



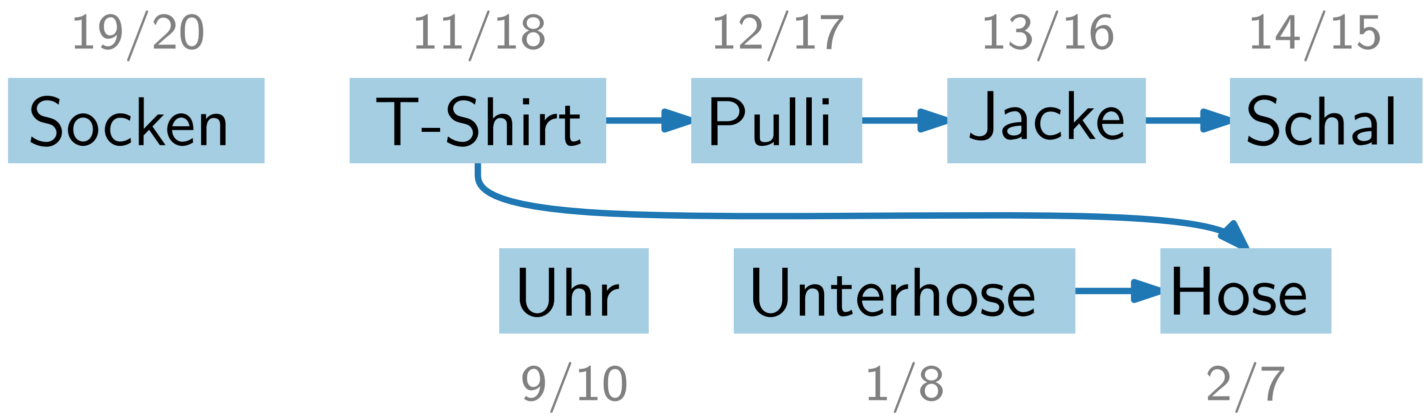
Idee: Nutze Tiefensuche!
Sortiere Knoten nach absteigenden f-Zeiten.



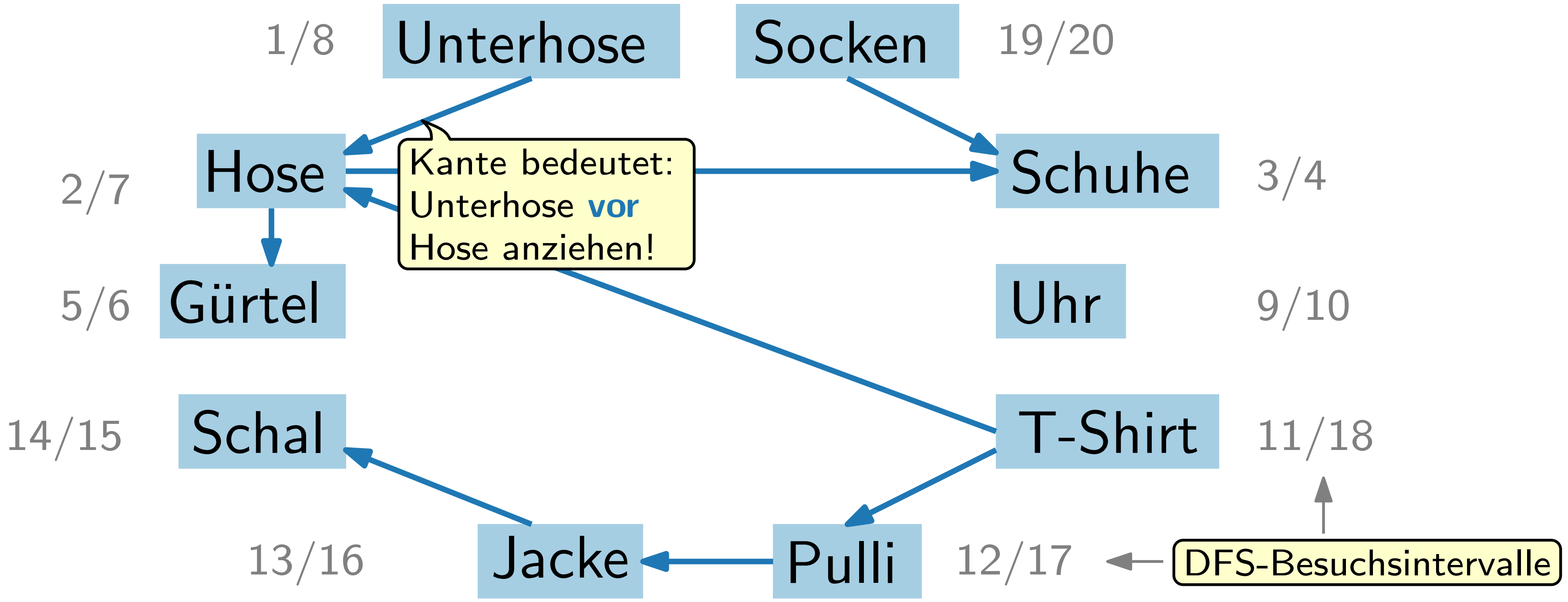
Ablaufplanung



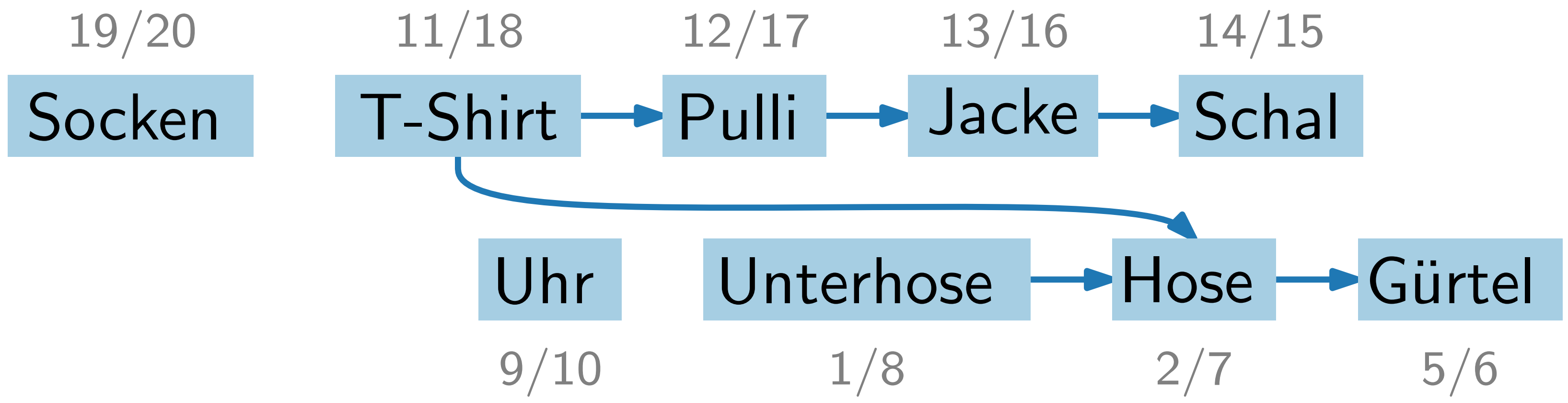
Idee: Nutze Tiefensuche!
Sortiere Knoten nach absteigenden f-Zeiten.



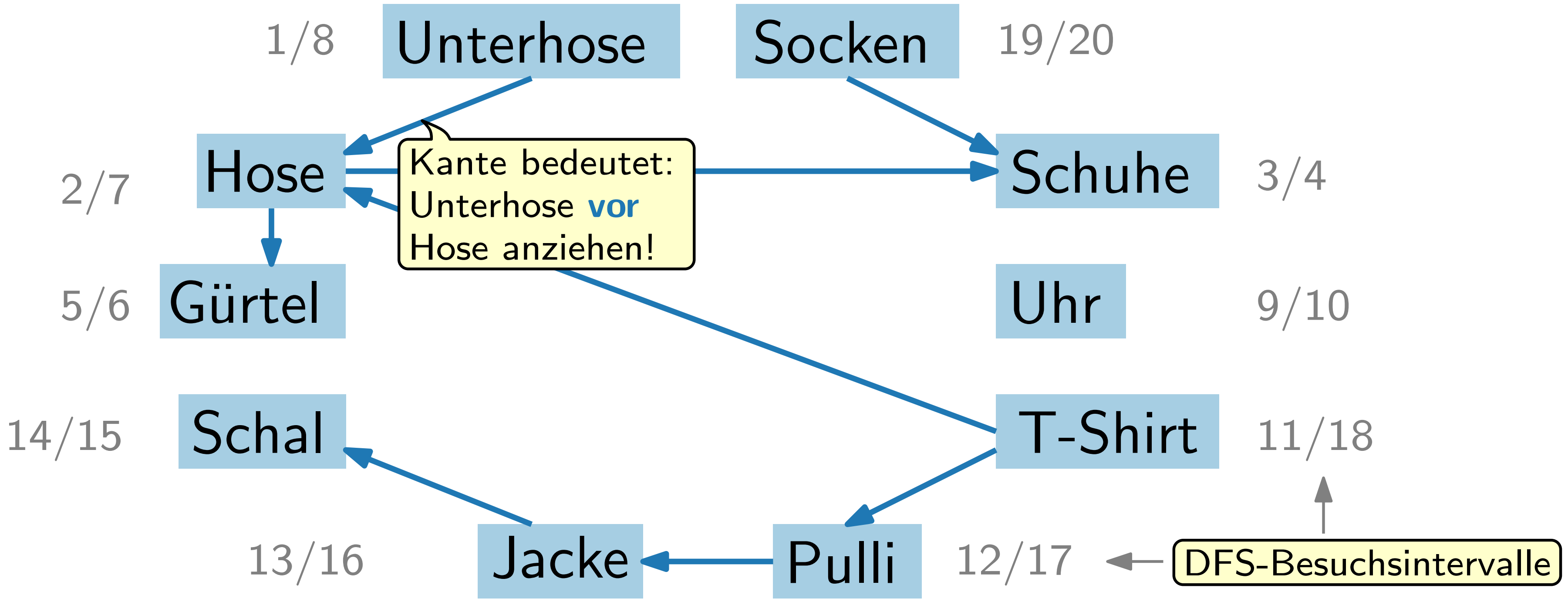
Ablaufplanung



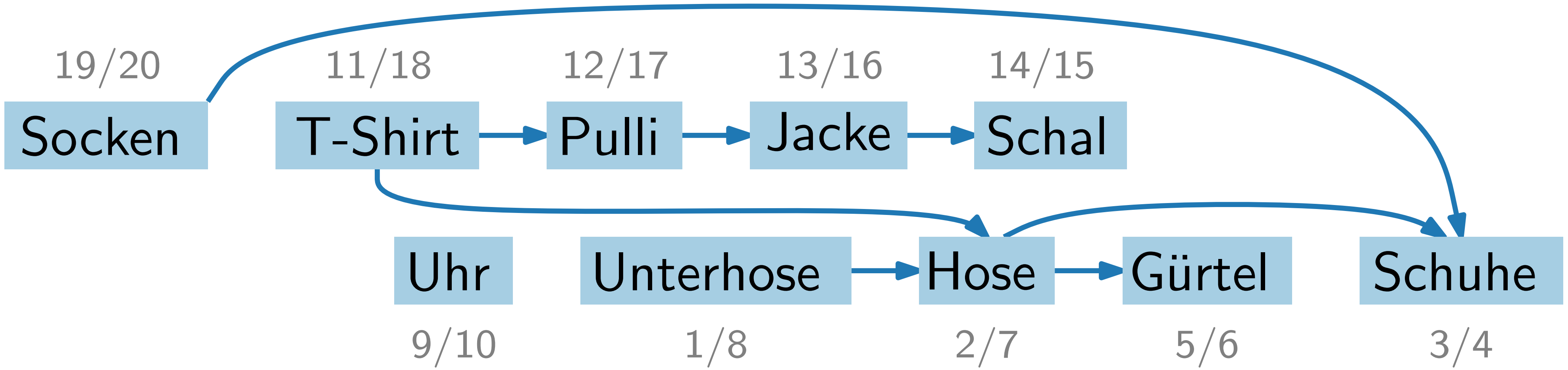
Idee: Nutze Tiefensuche!
Sortiere Knoten nach absteigenden f-Zeiten.



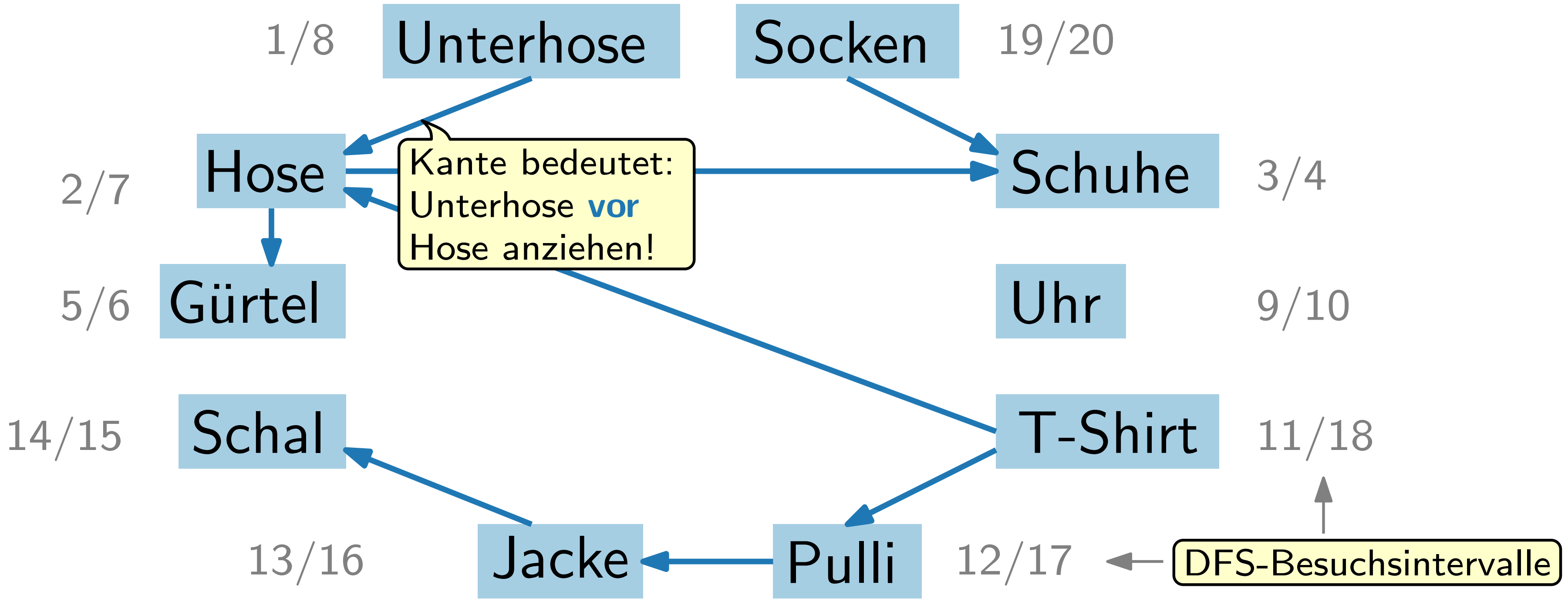
Ablaufplanung



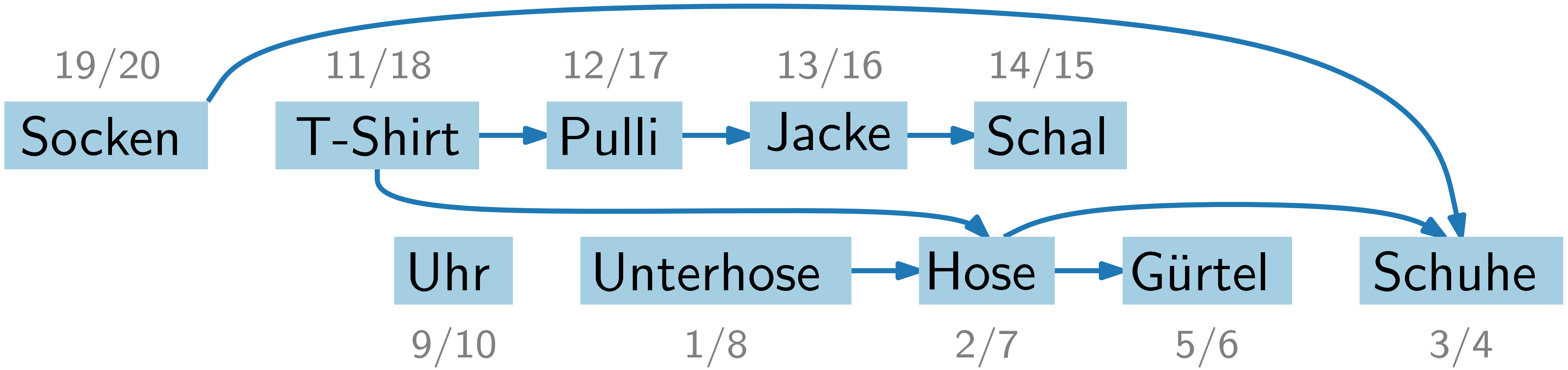
Idee: Nutze Tiefensuche!
Sortiere Knoten nach absteigenden f-Zeiten.



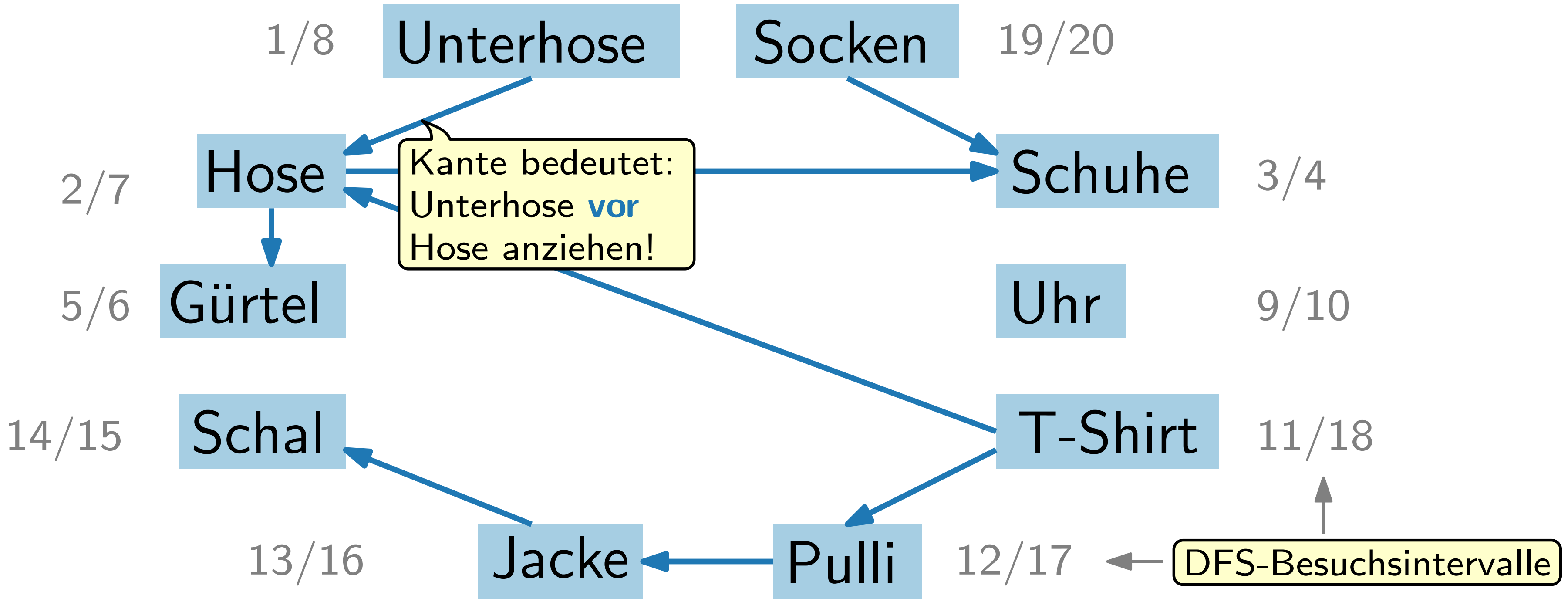
Ablaufplanung



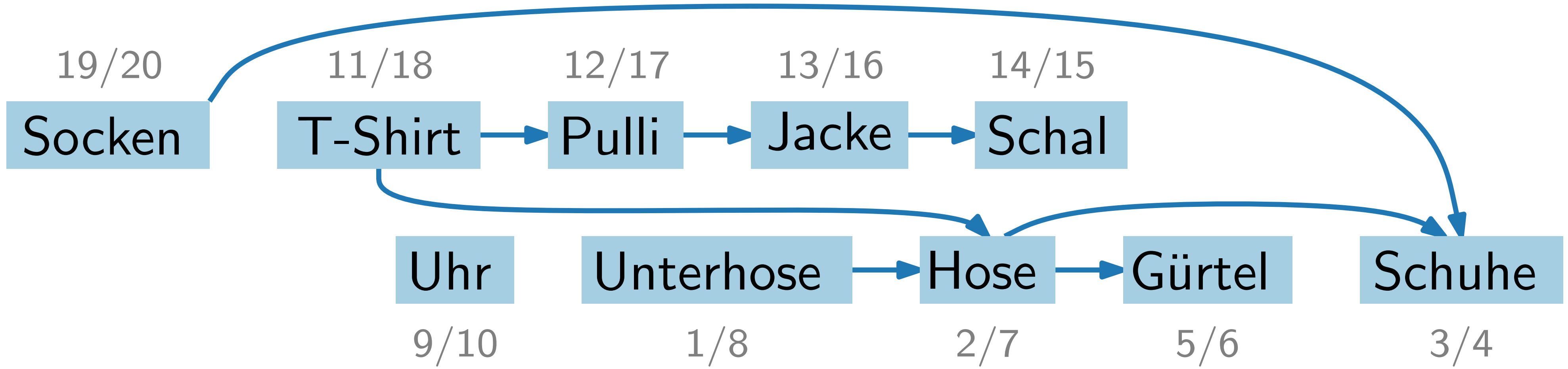
Idee: Nutze Tiefensuche! $\Rightarrow$ Alle Kanten sind ...
Sortiere Knoten nach absteigenden f-Zeiten.



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.

$\textsc{TopologicalSort}(\text{DirectedGraph } G)$
  $L = \textbf{new } \textsc{List}()$

Topologisch sortieren

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

```
TopologicalSort(DirectedGraph G)
  L = new List()
  DFS(G) mit folgender Änderung:
```

Topologisch sortieren

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

TopologicalSort(DirectedGraph G)
$L =$ **new** List()
DFS(G) mit folgender Änderung:

- Wenn ein Knoten blau gefärbt wird, häng ihn **vorne** an die Liste L an.

Topologisch sortieren

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

$\textsc{TopologicalSort}($DirectedGraph $G)$
  $L =$ **new** $\textsc{List}()$
  DFS(G) mit folgender Änderung:
  - Wenn ein Knoten blau 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 = **new** List()
  DFS(G) mit folgender Änderung:
  - Wenn ein Knoten blau 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.

$\textsc{TopologicalSort}$(DirectedGraph G)
  $L =$ **new** $\textsc{List}()$
  DFS(G) mit folgender Änderung:
  - Wenn ein Knoten blau gefärbt wird, häng ihn **vorne** an die Liste L an.

  return L

Laufzeit?

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

Topologisch sortieren

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

TopologicalSort(DirectedGraph G)
  L = **new** List()
  DFS(G) mit folgender Änderung:
  - Wenn ein Knoten blau gefärbt wird, häng ihn **vorne** an die Liste L an.

  return L

Laufzeit?
$\mathcal{O}(V + E)$

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)
  $L =$ **new** List()
  DFS(G) mit folgender Änderung:
  - Wenn ein Knoten blau gefärbt wird, häng ihn **vorne** an die Liste L an.

  return L

Laufzeit?
$\mathcal{O}(V + E)$

Korrekt?
Wann funktioniert's?

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

Topologisch sortieren

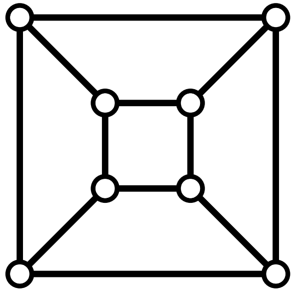
Topologische Sortierung: Lineare Ordnung der Knoten, so dass aus $(u, v) \in E$ folgt: u kommt vor v.

TopologicalSort(DirectedGraph G)
  L = **new** List()
  DFS(G) mit folgender Änderung:
  - Wenn ein Knoten blau gefärbt wird, häng ihn **vorne** an die Liste L an.

  return L

Laufzeit?
$\mathcal{O}(V + E)$

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.

$\textsc{TopologicalSort}(\text{DirectedGraph } G)$
- $L = $ **new** $\textsc{List}()$
- DFS(G) mit folgender Änderung:
  - Wenn ein Knoten blau gefärbt wird, häng ihn **vorne** an die Liste L an.
- **return** L

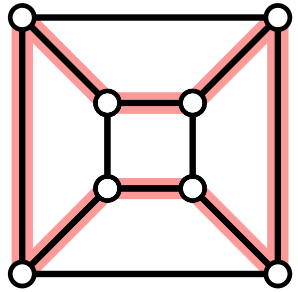
Laufzeit?

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

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.

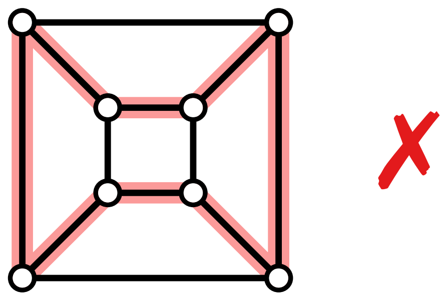
$\textsc{TopologicalSort}$(DirectedGraph G)
  $L = $ **new** $\textsc{List}()$
  DFS(G) mit folgender Änderung:
  - Wenn ein Knoten blau gefärbt wird, häng ihn **vorne** an die Liste L an.

  return L

Laufzeit?
$\mathcal{O}(V + E)$

Korrekt?
Wann funktioniert's?

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



Topologisch sortieren

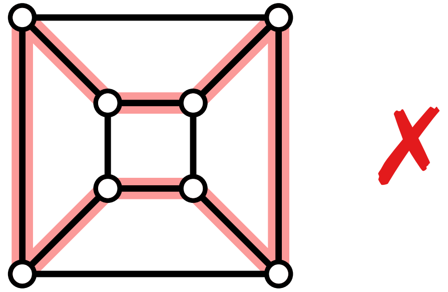
Topologische Sortierung: Lineare Ordnung der Knoten, so dass aus $(u, v) \in E$ folgt: u kommt vor v.

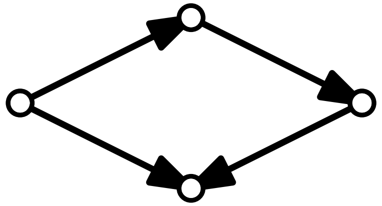
TopologicalSort(DirectedGraph G)
  L = **new** List()
  DFS(G) mit folgender Änderung:
  - Wenn ein Knoten blau gefärbt wird, häng ihn **vorne** an die Liste L an.

  return L

Laufzeit?
$\mathcal{O}(V + E)$

Korrekt?
Wann funktioniert's?





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

Topologisch sortieren

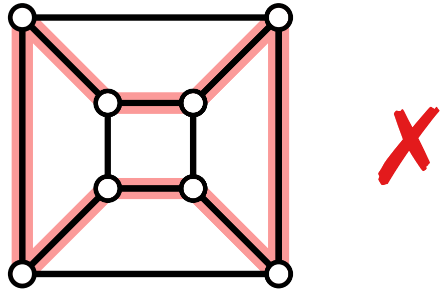
Topologische Sortierung: Lineare Ordnung der Knoten, so dass aus $(u, v) \in E$ folgt: u kommt vor v.

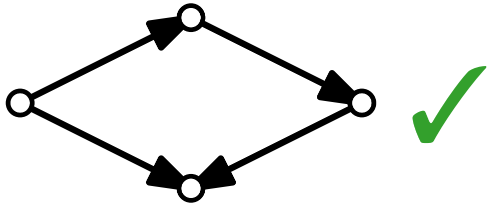
TopologicalSort(DirectedGraph G)
  L = **new** List()
  DFS(G) mit folgender Änderung:
  - Wenn ein Knoten blau gefärbt wird, häng ihn **vorne** an die Liste L an.

  return L

Laufzeit?
$\mathcal{O}(V + E)$

Korrekt?
Wann funktioniert's?





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

Kreisfrei $\Leftrightarrow$ keine R-Kanten

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

Kreisfrei $\Leftrightarrow$ keine R-Kanten

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

Beweis. „$\Rightarrow$“

„$\Leftarrow$“

Kreisfrei $\Leftrightarrow$ keine R-Kanten

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

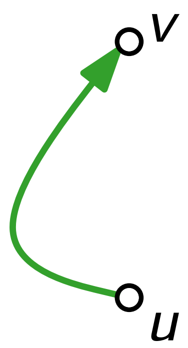
Beweis. „$\Rightarrow$“ Sei G kreisfrei.

„$\Leftarrow$“

Kreisfrei $\Leftrightarrow$ keine R-Kanten

Lemma. 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).

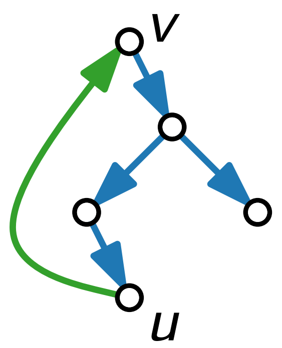


„$\Leftarrow$“

Kreisfrei $\Leftrightarrow$ keine R-Kanten

Lemma. 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.

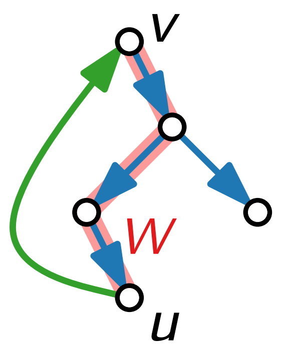


„$\Leftarrow$"

Kreisfrei $\Leftrightarrow$ keine R-Kanten

Lemma. 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.

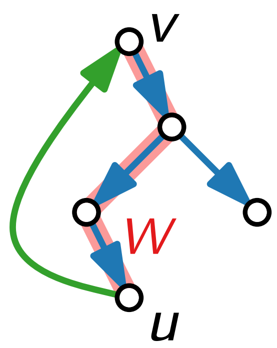
D.h. G enthält einen gerichteten v-u-Weg W.

„$\Leftarrow$“

Kreisfrei $\Leftrightarrow$ keine R-Kanten

Lemma. 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.

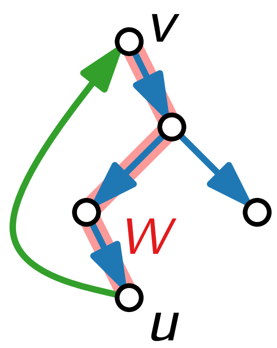
Aber dann ist $W \oplus (u, v)$ ein gerichteter Kreis.

„$\Leftarrow$"

Kreisfrei $\Leftrightarrow$ keine R-Kanten

Lemma. 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.

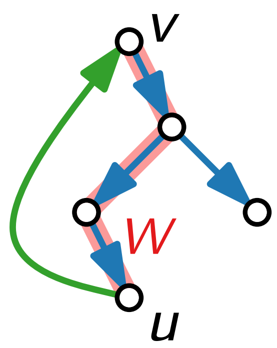
Aber dann ist $W \oplus (u, v)$ ein gerichteter Kreis. ↯

„$\Leftarrow$“

Kreisfrei $\Leftrightarrow$ keine R-Kanten

Lemma. 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.

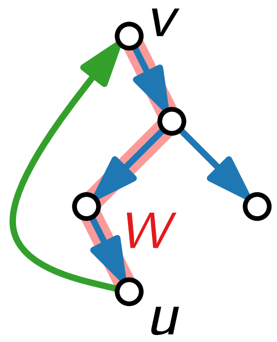
Aber dann ist $W \oplus (u, v)$ ein gerichteter Kreis. ↯

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

Kreisfrei $\Leftrightarrow$ keine R-Kanten

Lemma. 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).

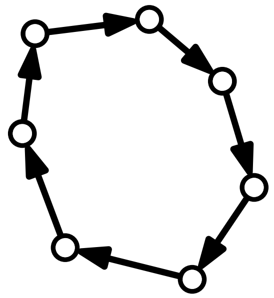
Dann ist u Nachfolger von v im DFS-Wald.

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.

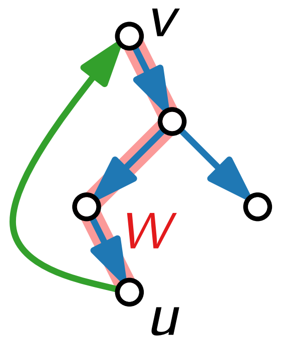
Ang. G enthält trotzdem Kreis $C = \langle v_1, \dots, v_k \rangle$.



Kreisfrei $\Leftrightarrow$ keine R-Kanten

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

Beweis. „$\Rightarrow$" Sei G kreisfrei.



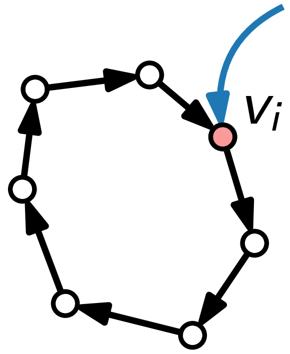
Angenommen DFS(G) liefert R-Kante (u, v).

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.



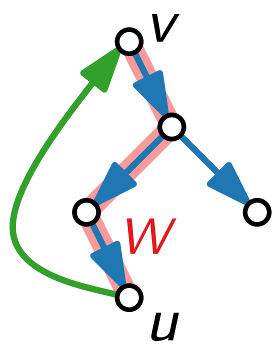
Ang. G enthält trotzdem Kreis $C = \langle v_1, \ldots, v_k \rangle$.

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

Kreisfrei $\Leftrightarrow$ keine R-Kanten

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

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



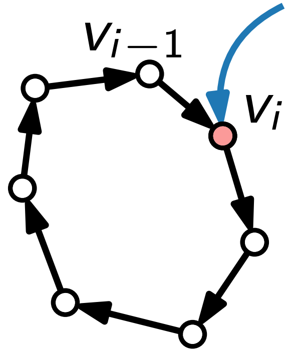
Angenommen DFS(G) liefert R-Kante (u, v).

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 $C = \langle v_1, \dots, v_k \rangle$.

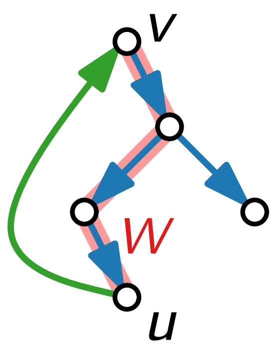
Sei v_i der 1. Knoten in C, den DFS(G) erreicht.

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

Kreisfrei $\Leftrightarrow$ keine R-Kanten

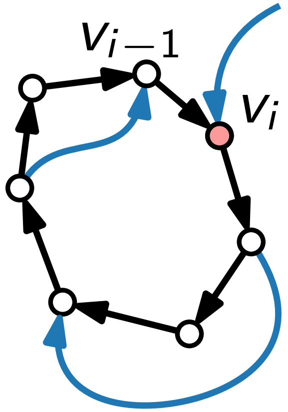
Lemma. Ein gerichteter Graph G ist kreisfrei
$\Leftrightarrow$ DFS(G) liefert keine Rückwärtskanten.

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.

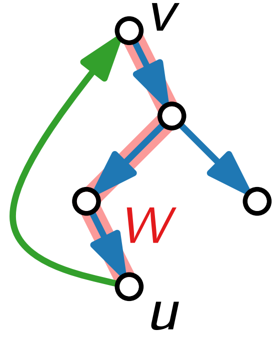


Ang. G enthält trotzdem Kreis $C = \langle v_1, \dots, v_k \rangle$.
Sei v_i der 1. Knoten in C, den DFS(G) erreicht.
Es gibt einen Weg von v_i nach v_{i-1} in G.
$\Rightarrow$ DFS gelangt zu v_{i-1}, solange v_i rot ist.

Kreisfrei $\Leftrightarrow$ keine R-Kanten

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

Beweis. „$\Rightarrow$" Sei G kreisfrei.



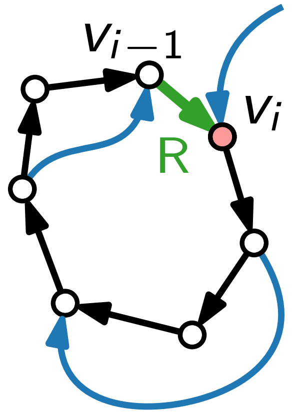
Angenommen DFS(G) liefert R-Kante (u, v).

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.

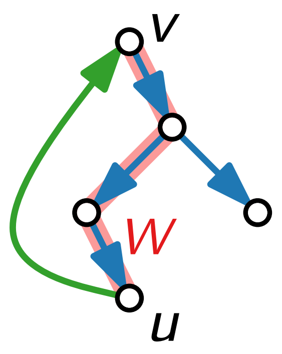
$\Rightarrow$ DFS gelangt zu v_{i-1}, solange v_i rot ist.

$\Rightarrow$ (v_{i-1}, v_i) ist R-Kante.

Kreisfrei $\Leftrightarrow$ keine R-Kanten

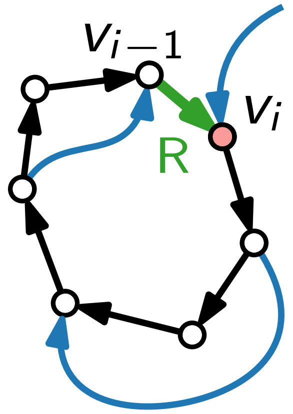
Lemma. Ein gerichteter Graph G ist kreisfrei
$\Leftrightarrow$ DFS(G) liefert keine Rückwärtskanten.

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 $C = \langle v_1, \ldots, v_k \rangle$.
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 rot ist.
$\Rightarrow$ (v_{i-1}, v_i) ist R-Kante. ⚡ $\square$

Korrektheit von TopologicalSort

Satz. Sei G ein gerichteter kreisfreier Graph. Dann liefert TopologicalSort(G) eine topologische Sortierung von G.

Korrektheit von TopologicalSort

Satz. Sei G ein gerichteter kreisfreier Graph. Dann liefert TopologicalSort(G) eine topologische Sortierung von G.

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

Korrektheit von TopologicalSort

Satz. Sei G ein gerichteter kreisfreier Graph. Dann liefert TopologicalSort(G) eine topologische Sortierung von G.

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

Dann gilt $v_n.f \quad \dots \qquad v_1.f$.

Korrektheit von TopologicalSort

Satz. Sei G ein gerichteter kreisfreier Graph. Dann liefert TopologicalSort(G) eine topologische Sortierung von G.

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

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

Korrektheit von TopologicalSort

Satz. Sei G ein gerichteter kreisfreier Graph. Dann liefert TopologicalSort(G) eine topologische Sortierung von G.

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

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

Korrektheit von TopologicalSort

Satz. Sei G ein gerichteter kreisfreier Graph. Dann liefert TopologicalSort(G) eine topologische Sortierung von G.

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

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

Sei (v_i, v_j) Kante von G.

Korrektheit von TopologicalSort

Satz. Sei G ein gerichteter kreisfreier Graph. Dann liefert TopologicalSort(G) eine topologische Sortierung von G.

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

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

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

Korrektheit von TopologicalSort

Satz. Sei G ein gerichteter kreisfreier Graph. Dann liefert TopologicalSort(G) eine topologische Sortierung von G.

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

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

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

Korrektheit von TopologicalSort

Satz. Sei G ein gerichteter kreisfreier Graph. Dann liefert TopologicalSort(G) eine topologische Sortierung von G.

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

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

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

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

Korrektheit von TopologicalSort

Satz. Sei G ein gerichteter kreisfreier Graph. Dann liefert TopologicalSort(G) eine topologische Sortierung von G.

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

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

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

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

- v_j rot

- v_j weiß

- v_j blau

Korrektheit von TopologicalSort

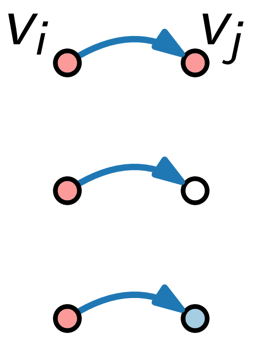
Satz. Sei G ein gerichteter kreisfreier Graph. Dann liefert TopologicalSort(G) eine topologische Sortierung von G.

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

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

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

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



- v_j rot $\Rightarrow$
- v_j weiß
- v_j blau

Korrektheit von TopologicalSort

Satz. Sei G ein gerichteter kreisfreier Graph. Dann liefert TopologicalSort(G) eine topologische Sortierung von G.

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

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

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

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

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

- v_j weiß

- v_j blau

Korrektheit von TopologicalSort

Satz. Sei G ein gerichteter kreisfreier Graph. Dann liefert TopologicalSort(G) eine topologische Sortierung von G.

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

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

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

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

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

- v_j weiß

- v_j blau

Korrektheit von TopologicalSort

Satz. Sei G ein gerichteter kreisfreier Graph. Dann liefert TopologicalSort(G) eine topologische Sortierung von G.

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

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

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

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

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

- v_j weiß

- v_j blau

Korrektheit von TopologicalSort

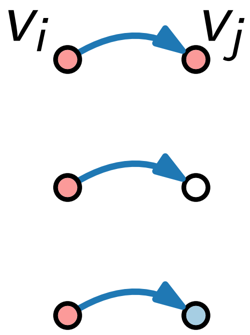
Satz. Sei G ein gerichteter kreisfreier Graph. Dann liefert $\textsc{TopologicalSort}(G)$ eine topologische Sortierung von G.

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

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

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

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



- v_j rot $\Rightarrow$ (v_i, v_j) ist R-Kante ↯ Widerspruch zu Lemma: G kreisfrei!
- v_j weiß $\Rightarrow$
- v_j blau

Korrektheit von TopologicalSort

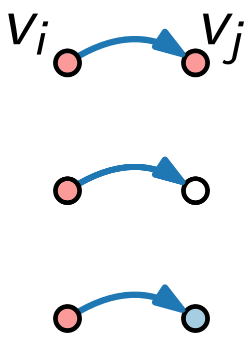
Satz. Sei G ein gerichteter kreisfreier Graph. Dann liefert TopologicalSort(G) eine topologische Sortierung von G.

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

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

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

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



- v_j rot $\Rightarrow$ (v_i, v_j) ist R-Kante ⚡ Widerspruch zu Lemma: G kreisfrei!
- v_j weiß $\Rightarrow$ v_j Nachfolger von v_i
- v_j blau

Korrektheit von TopologicalSort

Satz. Sei G ein gerichteter kreisfreier Graph. Dann liefert TopologicalSort(G) eine topologische Sortierung von G.

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

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

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

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

- v_j rot $\Rightarrow$ (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 blau

Korrektheit von TopologicalSort

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

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

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

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

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

- v_j rot $\Rightarrow$ (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 blau

Korrektheit von TopologicalSort

Satz. Sei G ein gerichteter kreisfreier Graph. Dann liefert TopologicalSort(G) eine topologische Sortierung von G.

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

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

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

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

- v_j rot $\Rightarrow$ (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 blau

Korrektheit von TopologicalSort

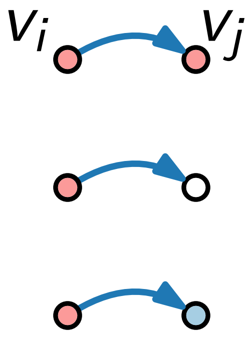
Satz. Sei G ein gerichteter kreisfreier Graph. Dann liefert TopologicalSort(G) eine topologische Sortierung von G.

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

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

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

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



- v_j rot $\Rightarrow$ (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 blau $\Rightarrow$

Korrektheit von TopologicalSort

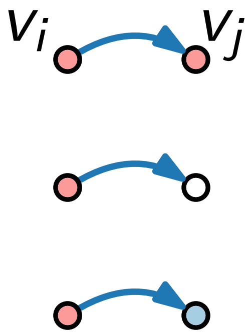
Satz. Sei G ein gerichteter kreisfreier Graph. Dann liefert TopologicalSort(G) eine topologische Sortierung von G.

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

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

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

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



- v_j rot $\Rightarrow$ (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 blau $\Rightarrow$ $v_i.f$ noch nicht gesetzt, $v_j.f$ gesetzt

Korrektheit von TopologicalSort

Satz. Sei G ein gerichteter kreisfreier Graph. Dann liefert TopologicalSort(G) eine topologische Sortierung von G.

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

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

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

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

- v_j rot $\Rightarrow$ (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 blau $\Rightarrow$ $v_i.f$ noch nicht gesetzt, $v_j.f$ gesetzt

  $\Rightarrow$

Korrektheit von TopologicalSort

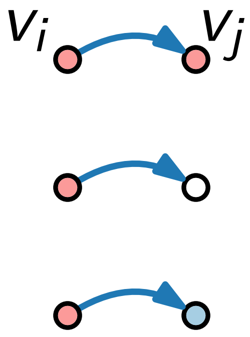
Satz. Sei G ein gerichteter kreisfreier Graph. Dann liefert TopologicalSort(G) eine topologische Sortierung von G.

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

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

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

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



- v_j rot $\Rightarrow$ (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 blau $\Rightarrow$ $v_i.f$ 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 TopologicalSort(G) eine topologische Sortierung von G.

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

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

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

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

- v_j rot $\Rightarrow$ (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 blau $\Rightarrow$ $v_i.f$ noch nicht gesetzt, $v_j.f$ gesetzt
  $\Rightarrow$ $v_i.f > v_j.f$ ✓

$\square$

Vergleich Durchlaufstrategien für Graphen

	Breitensuche	**Tiefensuche**
Laufzeit		
Ergebnis		
Datenstruktur		
Vorgehen		

Vergleich Durchlaufstrategien für Graphen

	Breitensuche	**Tiefensuche**
Laufzeit	$\mathcal{O}(V + E)$	
Ergebnis		
Datenstruktur		
Vorgehen		

Vergleich Durchlaufstrategien für Graphen

	Breitensuche	**Tiefensuche**
Laufzeit	$\mathcal{O}(V + E)$	$\mathcal{O}(V + E)$
Ergebnis		
Datenstruktur		
Vorgehen		

Vergleich Durchlaufstrategien für Graphen

	Breitensuche	**Tiefensuche**
Laufzeit	$\mathcal{O}(V+E)$	$\mathcal{O}(V+E)$
Ergebnis	BFS-Baum, d.h. kürzeste Wege	
Datenstruktur		
Vorgehen		

Vergleich Durchlaufstrategien für Graphen

	Breitensuche	**Tiefensuche**
Laufzeit	$\mathcal{O}(V + E)$	$\mathcal{O}(V + E)$
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	$\mathcal{O}(V + E)$	$\mathcal{O}(V + E)$
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	$\mathcal{O}(V+E)$	$\mathcal{O}(V+E)$
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	$\mathcal{O}(V + E)$	$\mathcal{O}(V + E)$
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	$\mathcal{O}(V + E)$	$\mathcal{O}(V + E)$
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