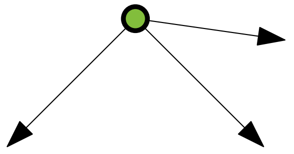
Karlsruher Institut für Technologie

Tutorium Algorithmen 1

09 · Tiefensuche · 24.6.2024
Peter Bohner Tutorium 3

Wiederholung: Tiefensuche

- Erkunde Graphen von einem Knoten aus
- Jeder Knoten wird (höchstens) ein Mal betrachtet
- Wir gehen nach Möglichkeit immer weiter



$visited \leftarrow \emptyset$

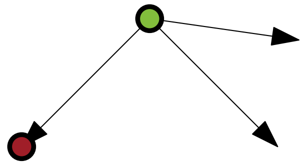
DFS(u)

foreach $v \in N(u)$ **do**
  if $v \notin visited$ **then**
    $visited.insert(v)$
    DFS(v)
  fi
od

Peter Bohner - peter.bohner@student.kit.edu

Wiederholung: Tiefensuche

- Erkunde Graphen von einem Knoten aus
- Jeder Knoten wird (höchstens) ein Mal betrachtet
- Wir gehen nach Möglichkeit immer weiter

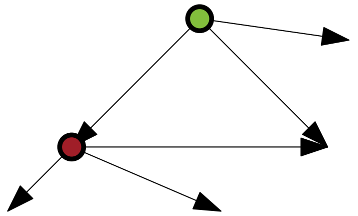


```
visited ← ∅
DFS(u)
  foreach v ∈ N(u) do
    if v ∉ visited then
        visited.insert(v)
        DFS(v)
    fi
  od
```


Peter Bohner - peter.bohner@student.kit.edu

Wiederholung: Tiefensuche

- Erkunde Graphen von einem Knoten aus
- Jeder Knoten wird (höchstens) ein Mal betrachtet
- Wir gehen nach Möglichkeit immer weiter



$visited \leftarrow \emptyset$

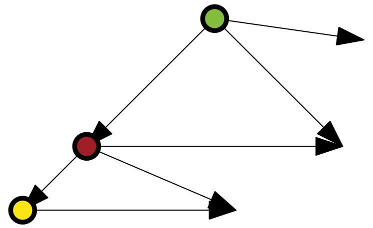
DFS(u)

foreach $v \in N(u)$ **do**
    if $v \notin visited$ **then**
        $visited.insert(v)$
        DFS(v)
    fi
od

Peter Bohner - peter.bohner@student.kit.edu

Wiederholung: Tiefensuche

- Erkunde Graphen von einem Knoten aus
- Jeder Knoten wird (höchstens) ein Mal betrachtet
- Wir gehen nach Möglichkeit immer weiter



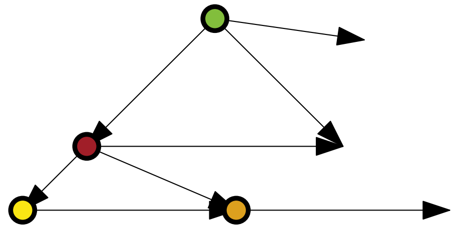
$visited \leftarrow \emptyset$

DFS(u)

> **foreach** $v \in N(u)$ **do**
>   **if** $v \notin visited$ **then**
> $visited.insert(v)$
>   **DFS**(v)
>   **fi**
> **od**

Wiederholung: Tiefensuche

- Erkunde Graphen von einem Knoten aus
- Jeder Knoten wird (höchstens) ein Mal betrachtet
- Wir gehen nach Möglichkeit immer weiter



$visited \leftarrow \emptyset$

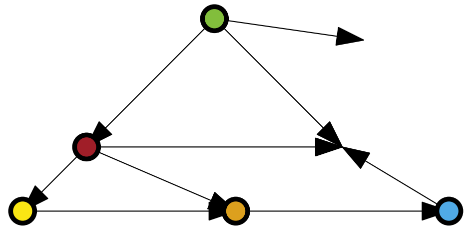
DFS(u)

> **foreach** $v \in N(u)$ **do**
>
>   **if** $v \notin visited$ **then**
>
> $visited.insert(v)$
>
>   **DFS**(v)
>
>   **fi**
>
> **od**

Peter Bohner - peter.bohner@student.kit.edu

Wiederholung: Tiefensuche

- Erkunde Graphen von einem Knoten aus
- Jeder Knoten wird (höchstens) ein Mal betrachtet
- Wir gehen nach Möglichkeit immer weiter



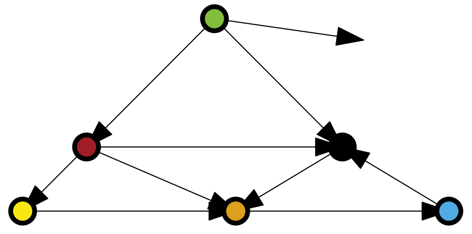
$visited \leftarrow \emptyset$

DFS(u)

foreach $v \in N(u)$ **do**
  if $v \notin visited$ **then**
    $visited.insert(v)$
    DFS(v)
  fi
od

Wiederholung: Tiefensuche

- Erkunde Graphen von einem Knoten aus
- Jeder Knoten wird (höchstens) ein Mal betrachtet
- Wir gehen nach Möglichkeit immer weiter



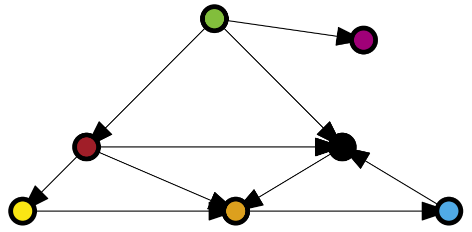
$visited \leftarrow \emptyset$

DFS(u)

```
foreach v ∈ N(u) do
    if v ∉ visited then
        visited.insert(v)
        DFS(v)
    fi
od
```

Wiederholung: Tiefensuche

- Erkunde Graphen von einem Knoten aus
- Jeder Knoten wird (höchstens) ein Mal betrachtet
- Wir gehen nach Möglichkeit immer weiter



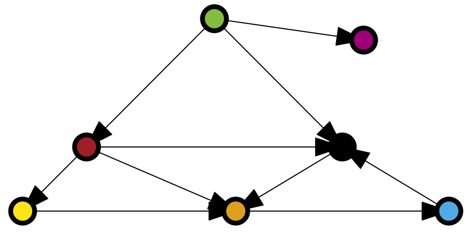
$visited \leftarrow \emptyset$

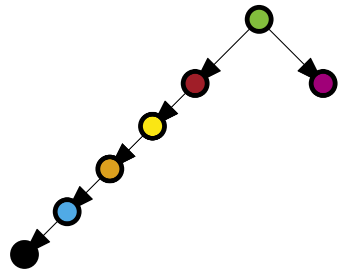
DFS(u)

foreach $v \in N(u)$ **do**
  if $v \notin visited$ **then**
    $visited.insert(v)$
    DFS(v)
  fi
od

Wiederholung: Tiefensuche

- Erkunde Graphen von einem Knoten aus
- Jeder Knoten wird (höchstens) ein Mal betrachtet
- Wir gehen nach Möglichkeit immer weiter
- Abgelaufene Kanten bilden Baum (Tiefensuchbaum)
  - Kanten heißen Baumkanten





$visited \leftarrow \emptyset$

DFS(u)

  foreach $v \in N(u)$ **do**

    if $v \notin visited$ **then**

      $visited.insert(v)$

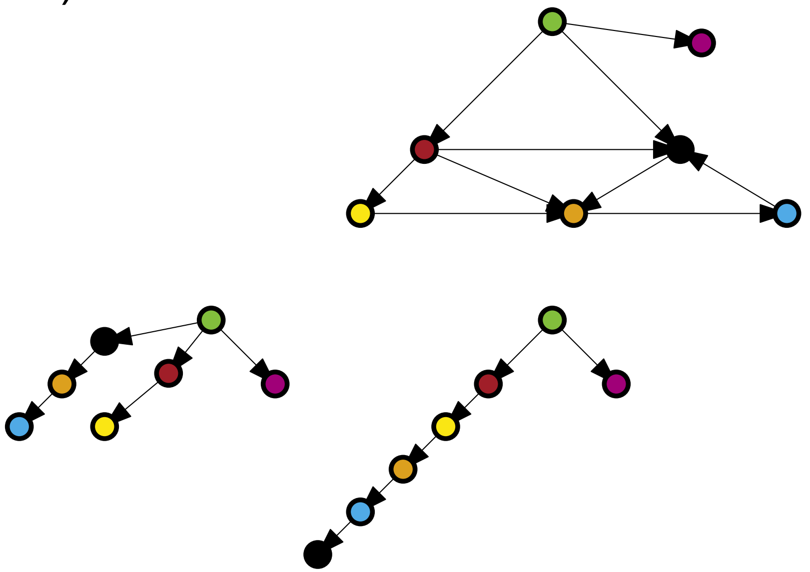
      DFS(v)

    fi

  od

Wiederholung: Tiefensuche

- Erkunde Graphen von einem Knoten aus
- Jeder Knoten wird (höchstens) ein Mal betrachtet
- Wir gehen nach Möglichkeit immer weiter
- Abgelaufene Kanten bilden Baum (Tiefensuchbaum)
  - Kanten heißen Baumkanten
- Tiefensuchbaum ist nicht eindeutig



$visited \leftarrow \emptyset$

DFS(u)

  foreach $v \in N(u)$ **do**

    if $v \notin visited$ **then**

      $visited.insert(v)$

      DFS(v)

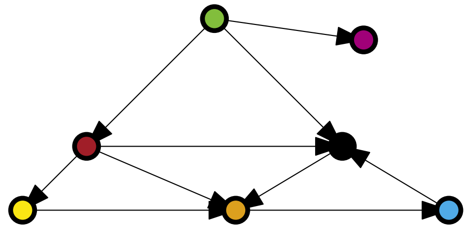
    fi

  od

Peter Bohner - peter.bohner@student.kit.edu

Wiederholung: Tiefensuche

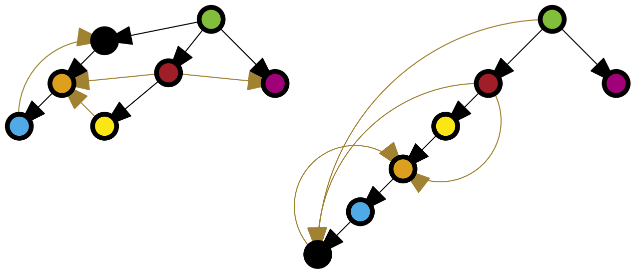
- Erkunde Graphen von einem Knoten aus
- Jeder Knoten wird (höchstens) ein Mal betrachtet
- Wir gehen nach Möglichkeit immer weiter
- Abgelaufene Kanten bilden Baum (Tiefensuchbaum)
  - Kanten heißen Baumkanten
- Tiefensuchbaum ist nicht eindeutig



$visited \leftarrow \emptyset$

DFS(u)

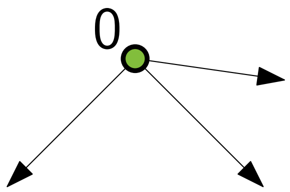
```
foreach v ∈ N(u) do
    if v ∉ visited then
        visited.insert(v)
        DFS(v)
    fi
od
```



wichtig: worst-case Laufzeit **immer** in $O(|V| + |E|)$

DFS/FIN-Nummer

- DFS-Nummer $\approx$ Entdeckungsreihenfolge
- Beispiel: Wurzel hat immer DFS-Nummer 0
- Kinder haben immer größere DFS-Nummer als Eltern

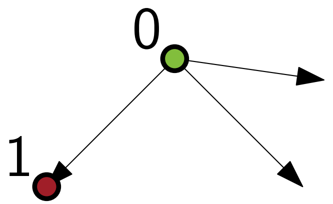


DFS-Baum

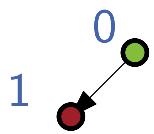
Peter Bohner - peter.bohner@student.kit.edu

DFS/FIN-Nummer

- DFS-Nummer ≈ Entdeckungsreihenfolge
- Beispiel: Wurzel hat immer DFS-Nummer 0
- Kinder haben immer größere DFS-Nummer als Eltern

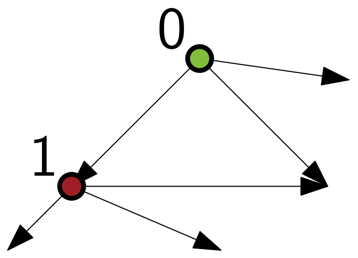


DFS-Baum

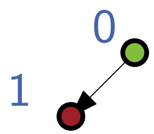


DFS/FIN-Nummer

- DFS-Nummer ≈ Entdeckungsreihenfolge
- Beispiel: Wurzel hat immer DFS-Nummer 0
- Kinder haben immer größere DFS-Nummer als Eltern

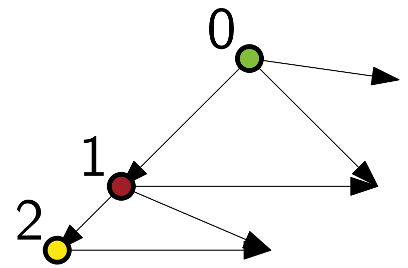


DFS-Baum

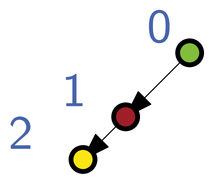


DFS/FIN-Nummer

- DFS-Nummer $\approx$ Entdeckungsreihenfolge
- Beispiel: Wurzel hat immer DFS-Nummer 0
- Kinder haben immer größere DFS-Nummer als Eltern



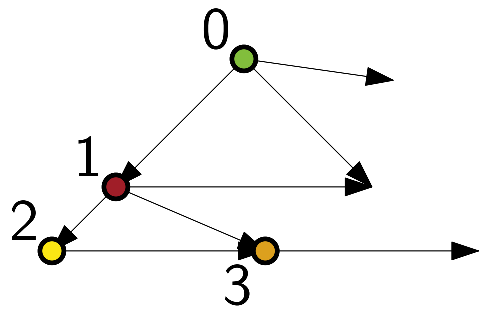
DFS-Baum



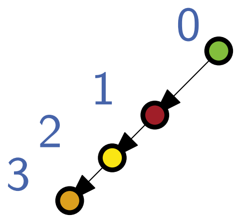
Peter Bohner - peter.bohner@student.kit.edu

DFS/FIN-Nummer

- DFS-Nummer ≈ Entdeckungsreihenfolge
- Beispiel: Wurzel hat immer DFS-Nummer 0
- Kinder haben immer größere DFS-Nummer als Eltern



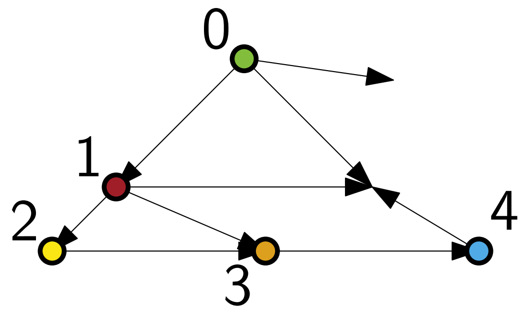
DFS-Baum



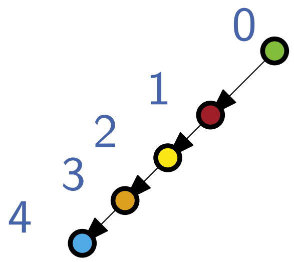
Peter Bohner - peter.bohner@student.kit.edu

DFS/FIN-Nummer

- DFS-Nummer ≈ Entdeckungsreihenfolge
- Beispiel: Wurzel hat immer DFS-Nummer 0
- Kinder haben immer größere DFS-Nummer als Eltern

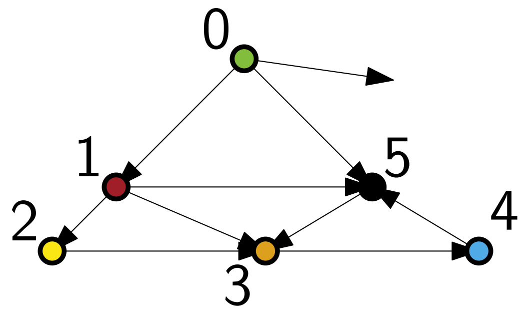


DFS-Baum

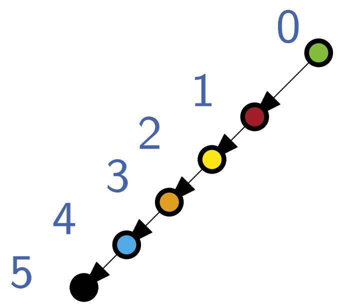


DFS/FIN-Nummer

- DFS-Nummer ≈ Entdeckungsreihenfolge
- Beispiel: Wurzel hat immer DFS-Nummer 0
- Kinder haben immer größere DFS-Nummer als Eltern

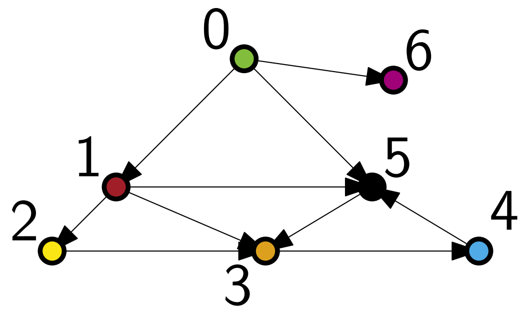


DFS-Baum

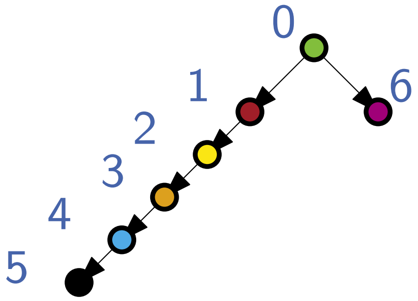


DFS/FIN-Nummer

- DFS-Nummer ≈ Entdeckungsreihenfolge
- Beispiel: Wurzel hat immer DFS-Nummer 0
- Kinder haben immer größere DFS-Nummer als Eltern

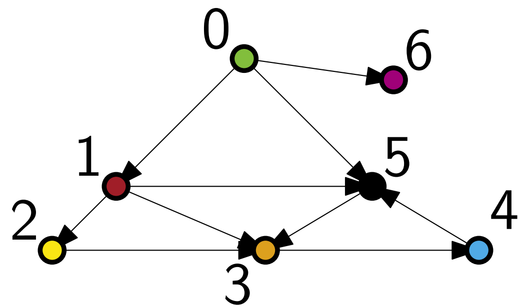


DFS-Baum

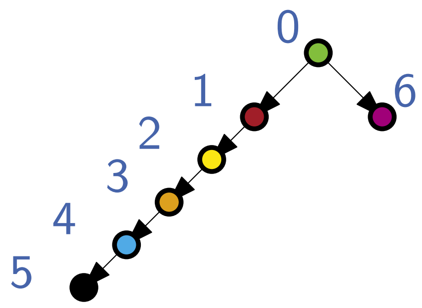


DFS/FIN-Nummer

- DFS-Nummer ≈ Entdeckungsreihenfolge
- Beispiel: Wurzel hat immer DFS-Nummer 0
- Kinder haben immer größere DFS-Nummer als Eltern
- FIN-Nummer ≈ backtrace-Reihenfolge
- FIN-Nummer = „Finish-Number“
- Reihenfolge, wann eigener Teilbaum vollständig erforscht wurde



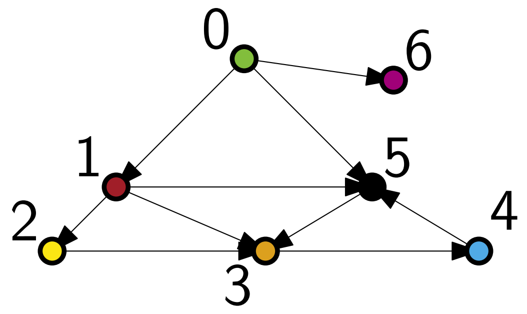
DFS-Baum



Peter Bohner - peter.bohner@student.kit.edu

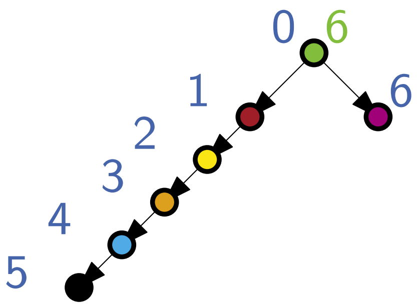
DFS/FIN-Nummer

- DFS-Nummer ≈ Entdeckungsreihenfolge
- Beispiel: Wurzel hat immer DFS-Nummer 0
- Kinder haben immer größere DFS-Nummer als Eltern
- FIN-Nummer ≈ backtrace-Reihenfolge
- FIN-Nummer = „Finish-Number“
- Reihenfolge, wann eigener Teilbaum vollständig erforscht wurde
  - Beispiel: Wurzel hat immer höchsten FIN-Nummer



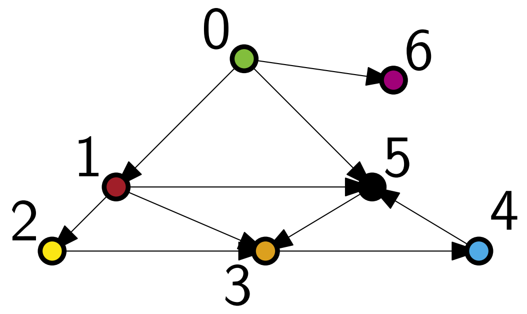
DFS-Baum

DFS-Nr./Fin-Nr.



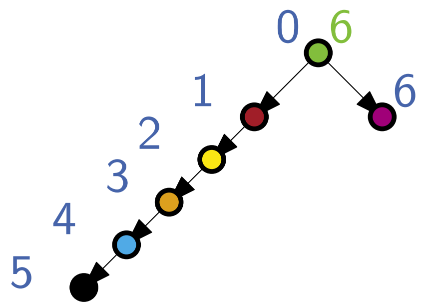
DFS/FIN-Nummer

- DFS-Nummer ≈ Entdeckungsreihenfolge
- Beispiel: Wurzel hat immer DFS-Nummer 0
- Kinder haben immer größere DFS-Nummer als Eltern
- FIN-Nummer ≈ backtrace-Reihenfolge
- FIN-Nummer = „Finish-Number“
- Reihenfolge, wann eigener Teilbaum vollständig erforscht wurde
  - Beispiel: Wurzel hat immer höchsten FIN-Nummer
- Kinder haben immer kleinere FIN-Nummer als Eltern



DFS-Baum

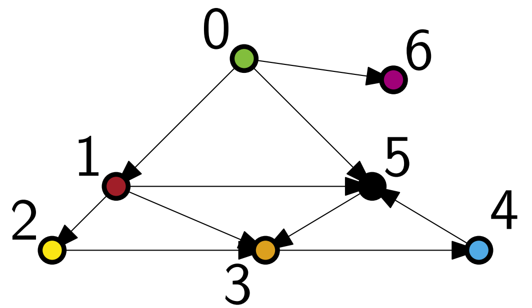
DFS-Nr./Fin-Nr.

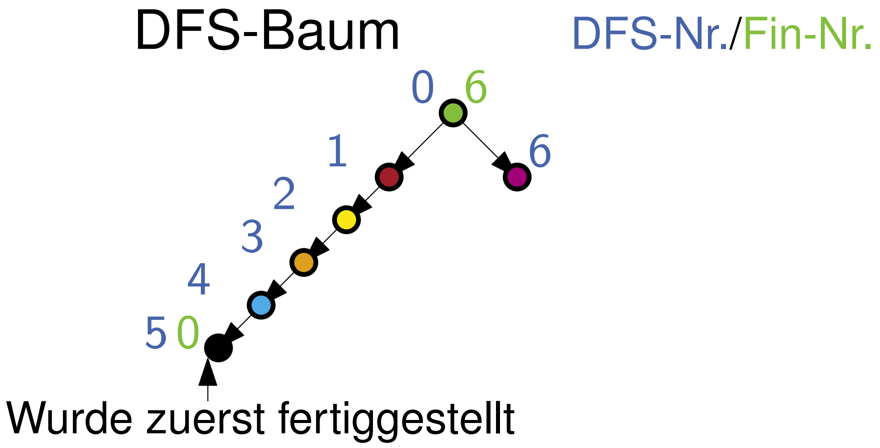


Peter Bohner - peter.bohner@student.kit.edu

DFS/FIN-Nummer

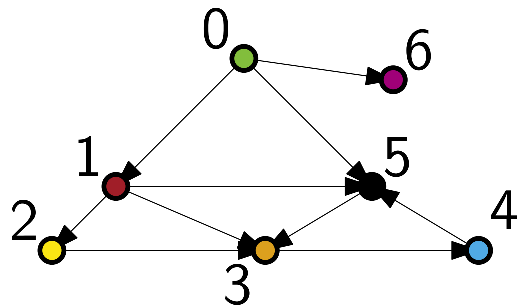
- DFS-Nummer ≈ Entdeckungsreihenfolge
- Beispiel: Wurzel hat immer DFS-Nummer 0
- Kinder haben immer größere DFS-Nummer als Eltern
- FIN-Nummer ≈ backtrace-Reihenfolge
- FIN-Nummer = „Finish-Number“
- Reihenfolge, wann eigener Teilbaum vollständig erforscht wurde
  - Beispiel: Wurzel hat immer höchsten FIN-Nummer
- Kinder haben immer kleinere FIN-Nummer als Eltern

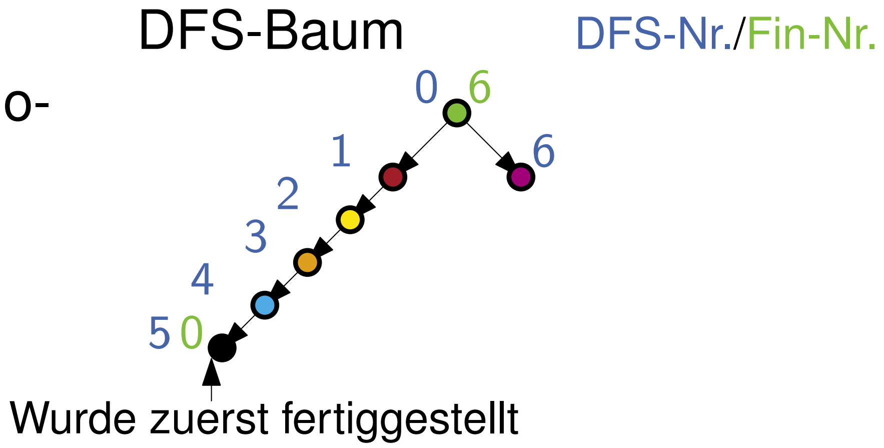




DFS/FIN-Nummer

- DFS-Nummer ≈ Entdeckungsreihenfolge
- Beispiel: Wurzel hat immer DFS-Nummer 0
- Kinder haben immer größere DFS-Nummer als Eltern
- FIN-Nummer ≈ backtrace-Reihenfolge
- FIN-Nummer = „Finish-Number“
- Reihenfolge, wann eigener Teilbaum vollständig erforscht wurde
  - Beispiel: Wurzel hat immer höchsten FIN-Nummer
- Kinder haben immer kleinere FIN-Nummer als Eltern
- Frage: Welche FIN-Nummern haben die anderen Knoten?

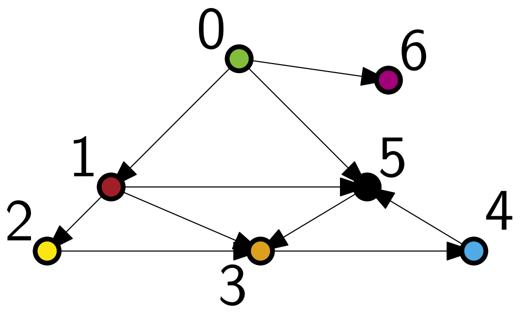




Peter Bohner - peter.bohner@student.kit.edu

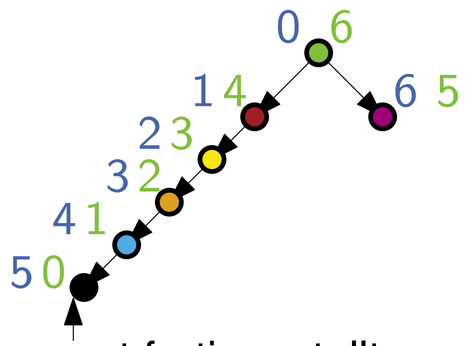
DFS/FIN-Nummer

- DFS-Nummer $\approx$ Entdeckungsreihenfolge
- Beispiel: Wurzel hat immer DFS-Nummer 0
- Kinder haben immer größere DFS-Nummer als Eltern
- FIN-Nummer $\approx$ backtrace-Reihenfolge
- FIN-Nummer = „Finish-Number“
- Reihenfolge, wann eigener Teilbaum vollständig erforscht wurde
  - Beispiel: Wurzel hat immer höchsten FIN-Nummer
- Kinder haben immer kleinere FIN-Nummer als Eltern
- Frage: Welche FIN-Nummern haben die anderen Knoten?



DFS-Baum

DFS-Nr./Fin-Nr.



Wurde zuerst fertiggestellt

Peter Bohner - peter.bohner@student.kit.edu

Aufgabe

Geben Sie eine DFS-Variante an, die jedem Knoten seine DFS-Nr. und FIN-Nr. zuweist (3 min)

$visited \leftarrow \emptyset$

DFS(u)

```
foreach v ∈ N(u) do
   if v ∉ visited then
      visited.insert(v)
      DFS(v)
   fi
od
```

Anweisung, um DFS-Nr. von u festzulegen: $dfsNb[u] \leftarrow \ldots$ (FIN analog)

Peter Bohner - peter.bohner@student.kit.edu

Aufgabe

Karlsruher Institut für Technologie

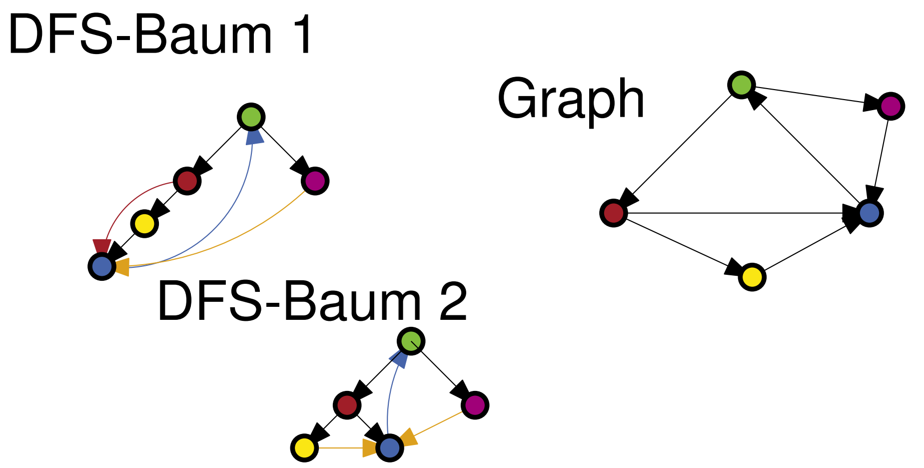
Geben Sie eine DFS-Variante an, die jedem Knoten seine DFS-Nr. und FIN-Nr. zuweist (3 min)

$visited \leftarrow \emptyset$
$nextDfsNb \leftarrow 0$
$nextFinNb \leftarrow 0$
DFS(u)
    $dfsNb[u] \leftarrow nextDfsNb{+}{+}$
    foreach $v \in N(u)$ **do**
        if $v \notin visited$ **then**
            $visited.insert(v)$
            DFS(v)
        fi
    od
    $finNb[u] \leftarrow nextFinNb{+}{+}$

Anweisung, um DFS-Nr. von u festzulegen: $dfsNb[u] \leftarrow \ldots$ (FIN analog)

Weitere Kantentypen

- zwei DFS-Bäume zu dem Graphen



- **rote** Kante heißt Vor(wärts)kante
- Kante (u, v) heißt Vorwärtskante $\Leftrightarrow$
  v ist von u über Baumkanten erreichbar
  $\wedge$ (u, v) ist nicht Baumkante
- überspringt quasi Knoten zwischen u und v

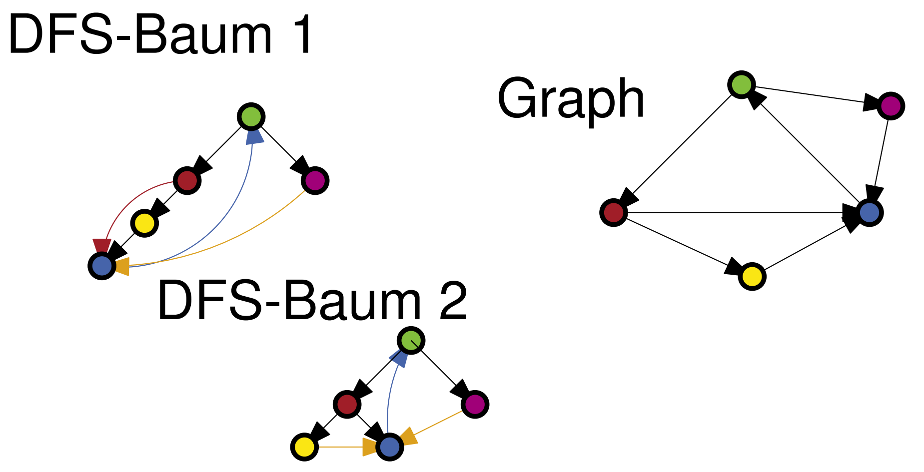
Peter Bohner - peter.bohner@student.kit.edu

Weitere Kantentypen

- zwei DFS-Bäume zu dem Graphen

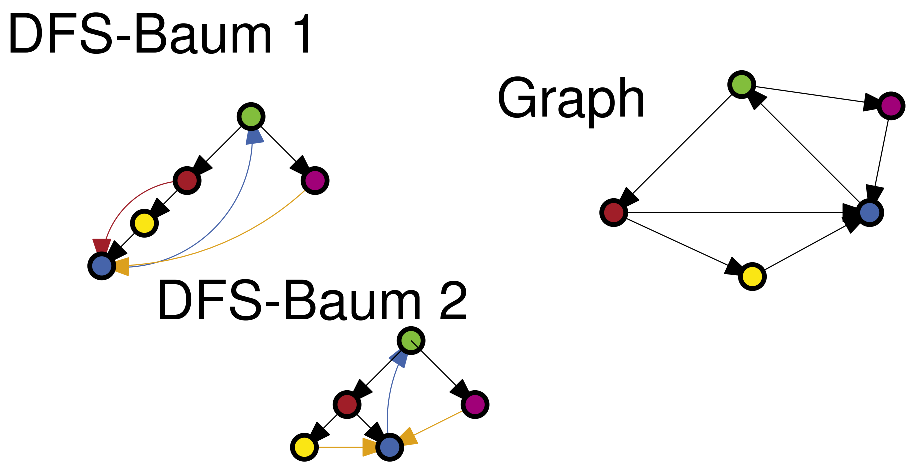
- **rote** Kante heißt Vor(wärts)kante
- Kante (u, v) heißt Vorwärtskante $\Leftrightarrow$
  v ist von u über Baumkanten erreichbar
  $\wedge$ (u, v) ist nicht Baumkante
- überspringt quasi Knoten zwischen u und v

- **orange** Kanten heißt Querkanten
- Kanten zwischen verschiedenen Teilbäumen



Weitere Kantentypen

- zwei DFS-Bäume zu dem Graphen



- **rote** Kante heißt Vor(wärts)kante
- Kante (u, v) heißt Vorwärtskante $\Leftrightarrow$
  v ist von u über Baumkanten erreichbar
  $\wedge$ (u, v) ist nicht Baumkante
- überspringt quasi Knoten zwischen u und v

- **orange** Kanten heißt Querkanten
- Kanten zwischen verschiedenen Teilbäumen

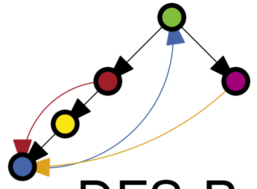
- **blaue** Kanten heißt Rück(wärts)kanten
- Kante (u, v) heißt Rückwärtskante $\Leftrightarrow$
  u ist von v über Baumkanten erreichbar $\Leftrightarrow$
  „v ist über u im DFS-Baum“

Zyklen

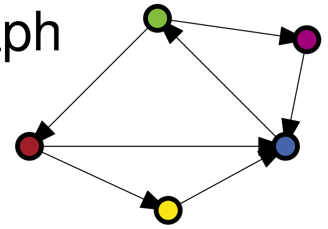
- Kante (u, v) heißt Rückwärtskante $\Leftrightarrow$
  u ist von v über Baumkanten erreichbar $\Leftrightarrow$
  „v ist über u im DFS-Baum“

$\Rightarrow$ Mit jeder Rückkante lässt sich Zyklus konstruieren

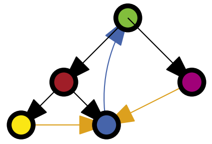
DFS-Baum 1



Graph



DFS-Baum 2



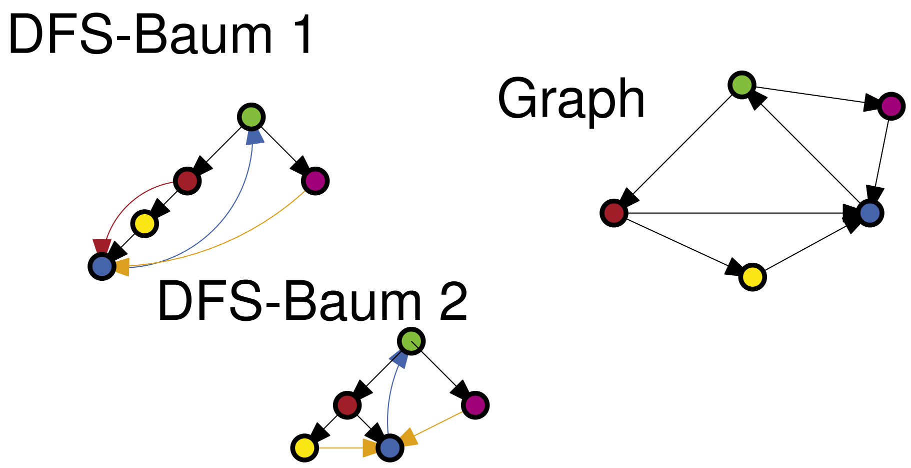
Zyklen

- Kante (u, v) heißt Rückwärtskante $\Leftrightarrow$
  u ist von v über Baumkanten erreichbar $\Leftrightarrow$
  „v ist über u im DFS-Baum“

$\Rightarrow$ Mit jeder Rückkante lässt sich Zyklus konstruieren

- Jeder Zyklus führt über min. eine Rückkante

Das gilt für jeden DFS-Baum, in dem u (bzw. v) vorkommt



Zyklen

- Kante (u, v) heißt Rückwärtskante $\Leftrightarrow$
  u ist von v über Baumkanten erreichbar $\Leftrightarrow$
  „v ist über u im DFS-Baum“

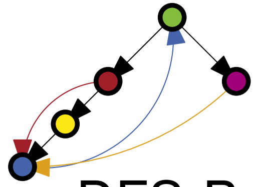
$\Rightarrow$ Mit jeder Rückkante lässt sich Zyklus konstruieren

- Jeder Zyklus führt über min. eine Rückkante

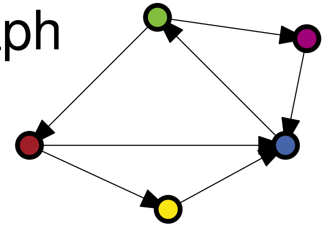
Das gilt für jeden DFS-Baum, in dem u (bzw. v) vorkommt

- Welche Kanten des Zyklus Rückkanten sind, ist abhängig von Baum

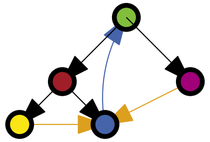
DFS-Baum 1



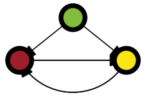
Graph

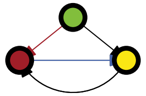


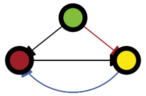
DFS-Baum 2



Graph







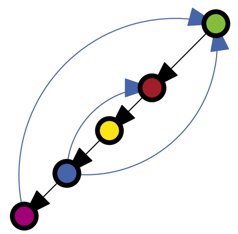
- Lila wurde zuerst erkundet
- Rot wurde zuerst erkundet

Peter Bohner - peter.bohner@student.kit.edu

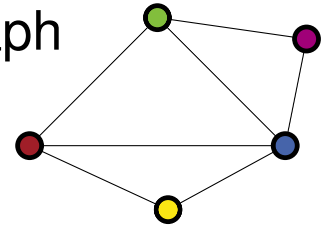
DFS in ungerichteten Graphen

- Unterschied: Kanten bekommen beim Betrachten Richtung

DFS-Baum 1



Graph



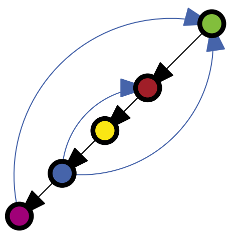
Peter Bohner - peter.bohner@student.kit.edu

DFS in ungerichteten Graphen

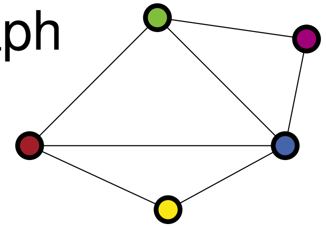
- Unterschied: Kanten bekommen beim Betrachten Richtung

Frage: Können hier Vorkanten auftreten?

DFS-Baum 1



Graph



DFS in ungerichteten Graphen

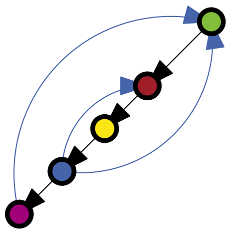
- Unterschied: Kanten bekommen beim Betrachten Richtung

Frage: Können hier Vorkanten auftreten?

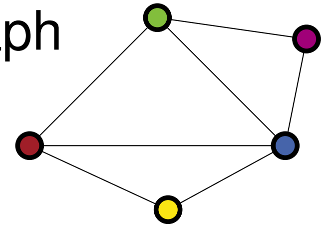
- Damit (u, v) Vorkante wird, muss v vollständig erkundet sein

$\Rightarrow$ Kante (v, u) wurde bereits betrachtet
$\Rightarrow$ Kante (u, v) existiert nicht mehr

DFS-Baum 1



Graph



DFS in ungerichteten Graphen

- Unterschied: Kanten bekommen beim Betrachten Richtung

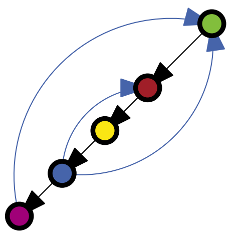
Frage: Können hier Vorkanten auftreten?

- Damit (u, v) Vorkante wird, muss v vollständig erkundet sein

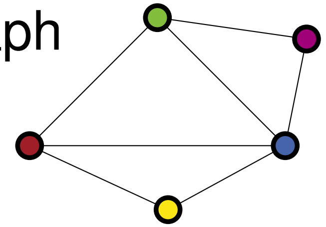
$\Rightarrow$ Kante (v, u) wurde bereits betrachtet
$\Rightarrow$ Kante (u, v) existiert nicht mehr

Frage: Können hier Querkanten auftreten?
$\Rightarrow$ Nein (gleiches Argument wie oben)

DFS-Baum 1



Graph



Peter Bohner - peter.bohner@student.kit.edu

DFS in ungerichteten Graphen

- Unterschied: Kanten bekommen beim Betrachten Richtung

Frage: Können hier Vorkanten auftreten?

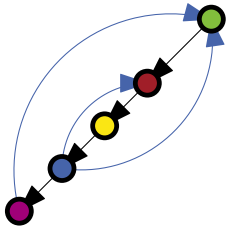
- Damit (u, v) Vorkante wird, muss v vollständig erkundet sein

$\Rightarrow$ Kante (v, u) wurde bereits betrachtet
$\Rightarrow$ Kante (u, v) existiert nicht mehr

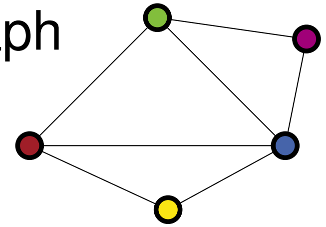
Frage: Können hier Querkanten auftreten?
$\Rightarrow$ Nein (gleiches Argument wie oben)

- Rückkanten bilden wieder Kreise
- Baumkanten bilden wieder Baum

DFS-Baum 1



Graph



Peter Bohner - peter.bohner@student.kit.edu

DFS in ungerichteten Graphen

- Unterschied: Kanten bekommen beim Betrachten Richtung

Frage: Können hier Vorkanten auftreten?

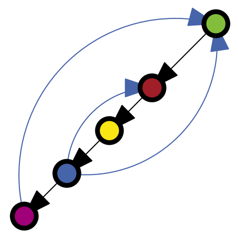
- Damit (u, v) Vorkante wird, muss v vollständig erkundet sein

$\Rightarrow$ Kante (v, u) wurde bereits betrachtet
$\Rightarrow$ Kante (u, v) existiert nicht mehr

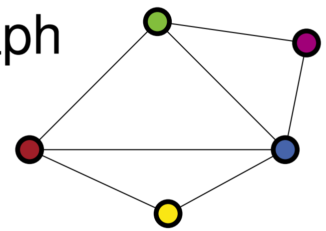
Frage: Können hier Querkanten auftreten?
$\Rightarrow$ Nein (gleiches Argument wie oben)

- Rückkanten bilden wieder Kreise
- Baumkanten bilden wieder Baum

DFS-Baum 1



Graph



Geben Sie eine DFS-Variante in Pseudocode an, die für einen ungerichteten Graphen für jede Baumkante die Funktion **treeEdge**(e) und für jede Rückkante die Funktion **backEdge**(e) ausführt (3 min)

Peter Bohner - peter.bohner@student.kit.edu

Aufgabe

Karlsruher Institut für Technologie

Geben Sie eine DFS-Variante in Pseudocode an, die für einen ungerichteten Graphen für jede Baumkante die Funktion **treeEdge**(e) und für jede Rückkante die Funktion **backEdge**(e) ausführt (3 min)

Aufgabe

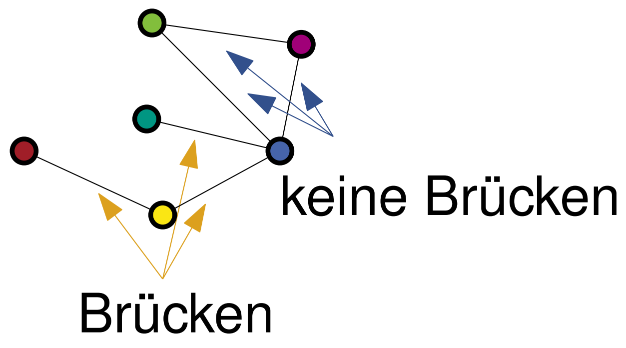
Karlsruher Institut für Technologie

Geben Sie eine DFS-Variante in Pseudocode an, die für einen ungerichteten Graphen für jede Baumkante die Funktion **treeEdge**(e) und für jede Rückkante die Funktion **backEdge**(e) ausführt (3 min)

```
visited ← ∅
nextDfsNb ← 0
DFS(u, parent = ⊥)
  dfsNb[u] ← nextDfsNb++
  foreach v ∈ N(u) do
     if v ∉ visited then
        visited.insert(v)
        treeEdge(u, v)
        DFS(v, u)
    else if dfsNb[v] < dfsNb[u] ∧ parent ≠ v then
        backEdge(u, v)
     fi
  od
```

Brücken in ungerichteten Graphen

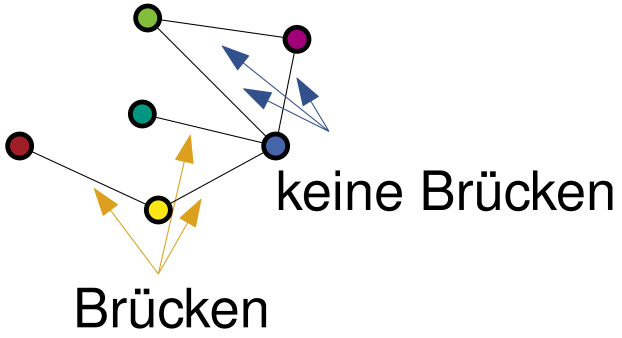
Kante $\{u, v\}$ heißt Brücke $\Leftrightarrow$In $G' = (V, E \setminus \{\{u, v\}\})$ ist v nicht von u erreichbar



Brücken in ungerichteten Graphen

Kante $\{u, v\}$ heißt Brücke $\Leftrightarrow$In $G' = (V, E \setminus \{\{u, v\}\})$ ist v nicht von u erreichbar

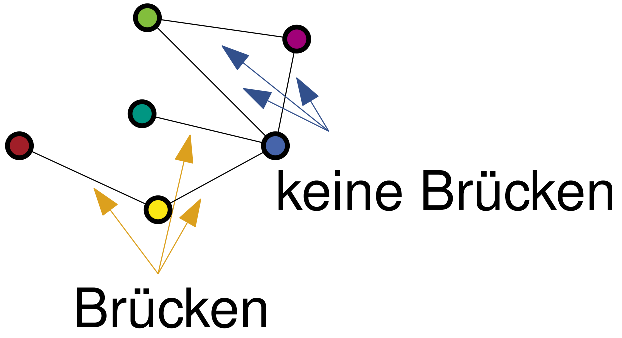
$\Leftrightarrow \forall$ Zyklus Z in G : $\{u, v\}$ ist nicht in Z



Brücken in ungerichteten Graphen

Kante $\{u, v\}$ heißt Brücke $\Leftrightarrow$In $G' = (V, E \setminus \{\{u, v\}\})$ ist v nicht von u erreichbar
$\Leftrightarrow \forall$ Zyklus Z in G : $\{u, v\}$ ist nicht in Z

o.B.d.A. ist G zusammenhängend
Frage: Warum ist das bei diesem Problem möglich?

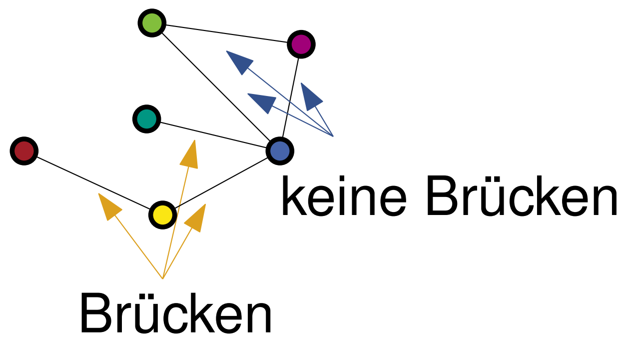


Peter Bohner - peter.bohner@student.kit.edu

Brücken in ungerichteten Graphen

Kante $\{u, v\}$ heißt Brücke $\Leftrightarrow$In $G' = (V, E \setminus \{\{u, v\}\})$ ist v nicht von u erreichbar
$\Leftrightarrow \forall$ Zyklus Z in G : $\{u, v\}$ ist nicht in Z

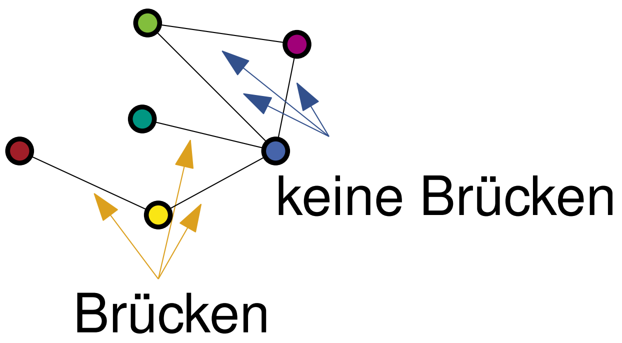
o.B.d.A. ist G zusammenhängend
Frage: Warum ist das bei diesem Problem möglich?
Nur die eigene Zusammenhangskomponente ist interessant.



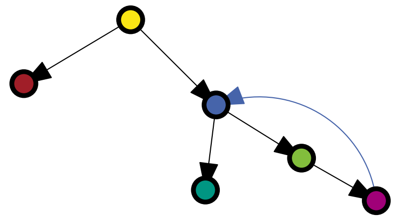
Brücken in ungerichteten Graphen

Kante $\{u, v\}$ heißt Brücke $\Leftrightarrow$In $G' = (V, E \setminus \{\{u, v\}\})$ ist v nicht von u erreichbar
$\Leftrightarrow \forall$ Zyklus Z in G : $\{u, v\}$ ist nicht in Z

o.B.d.A. ist G zusammenhängend
Frage: Warum ist das bei diesem Problem möglich?
Nur die eigene Zusammenhangskomponente ist interessant.



Idee: Betrachte (beliebigen) DFS-Graphen



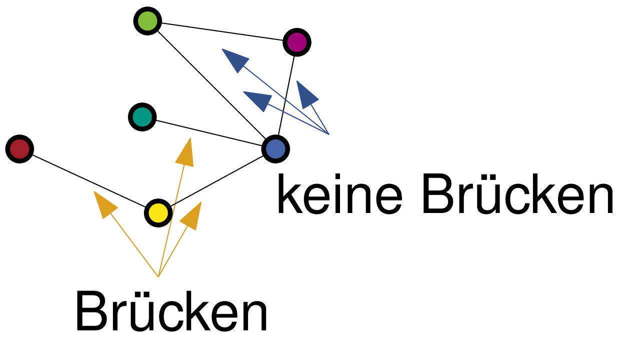
Frage: Können Rückkanten Brücken sein?

Peter Bohner - peter.bohner@student.kit.edu

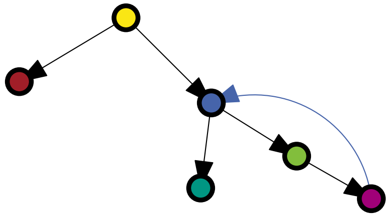
Brücken in ungerichteten Graphen

Kante $\{u, v\}$ heißt Brücke $\Leftrightarrow$In $G' = (V, E \setminus \{\{u, v\}\})$ ist v nicht von u erreichbar
$\Leftrightarrow \forall$ Zyklus Z in G : $\{u, v\}$ ist nicht in Z

o.B.d.A. ist G zusammenhängend
Frage: Warum ist das bei diesem Problem möglich?
Nur die eigene Zusammenhangskomponente ist interessant.



Idee: Betrachte (beliebigen) DFS-Graphen



Frage: Können Rückkanten Brücken sein?

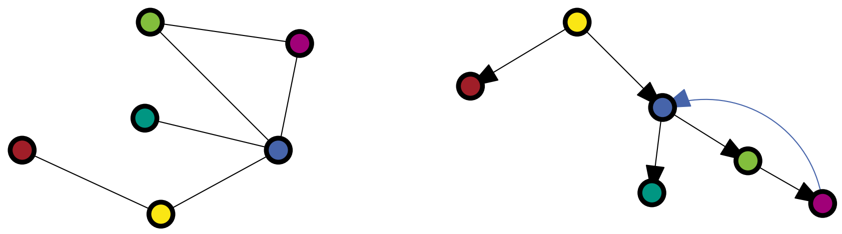
- u und v sind bereits über Baumkanten verbunden

lowNode

Wir definieren **lowNode**(v) =„Höchster Knoten, der von einem Knoten im Teilbaum von v direkt erreichbar ist"

$\{u, v\}$ ist Brücke $\Leftrightarrow$ **lowNode**(v) ist niedriger als u

Beispiel: **lowNode**$(\bullet) = \bullet$, **lowNode**$(\bullet) = \bullet$, **lowNode**$(\bullet) = \bullet$

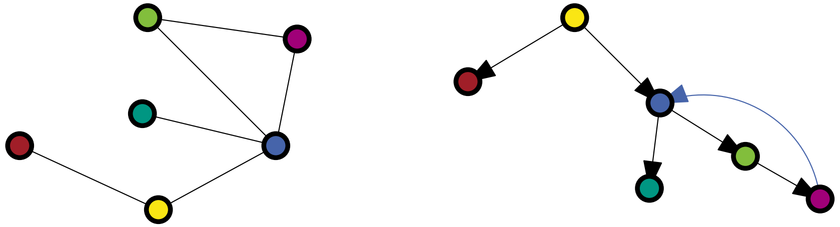


lowNode

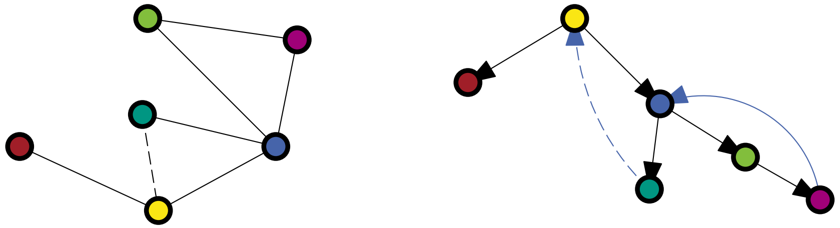
Wir definieren **lowNode**(v) =„Höchster Knoten, der von einem Knoten im Teilbaum von v direkt erreichbar ist“

$\{u, v\}$ ist Brücke $\Leftrightarrow$ **lowNode**(v) ist niedriger als u

Beispiel: **lowNode**$(\bullet) = \bullet$, **lowNode**$(\bullet) = \bullet$, **lowNode**$(\bullet) = \bullet$



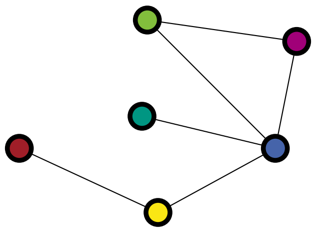
Beispiel: **lowNode**$(\bullet) = \bullet$, **lowNode**$(\bullet) = \bullet$, **lowNode**$(\bullet) = \bullet$



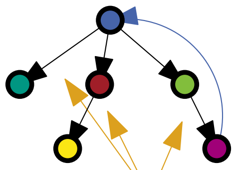
Peter Bohner - peter.bohner@student.kit.edu

DFS-Nr. für ungerichtete Graphen

- verschiedene Teilbäume sind immer isoliert



$\Rightarrow$

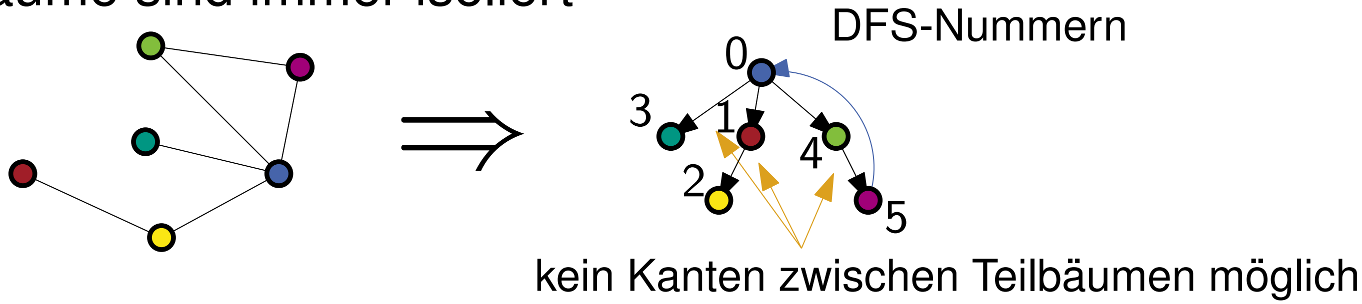


kein Kanten zwischen Teilbäumen möglich

Peter Bohner - peter.bohner@student.kit.edu

DFS-Nr. für ungerichtete Graphen

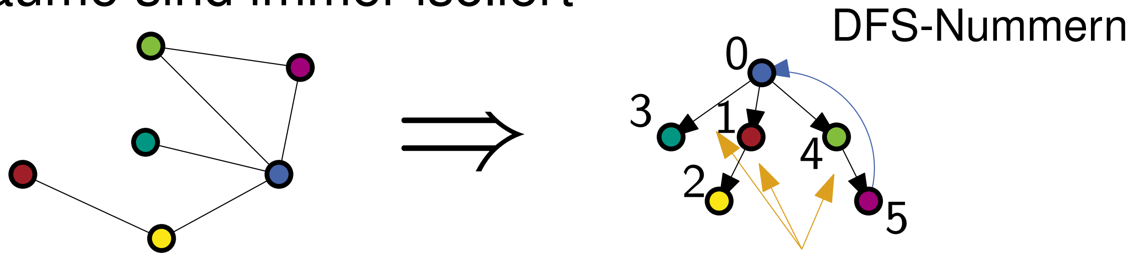
- verschiedene Teilbäume sind immer isoliert



⇒ Für jede Kante gibt die DFS-Nr. an, welcher Knoten höher liegt

DFS-Nr. für ungerichtete Graphen

- verschiedene Teilbäume sind immer isoliert



kein Kanten zwischen Teilbäumen möglich

$\Rightarrow$ Für jede Kante gibt die DFS-Nr. an, welcher Knoten höher liegt

$\Rightarrow$ ersetze **lowNode** durch **low**

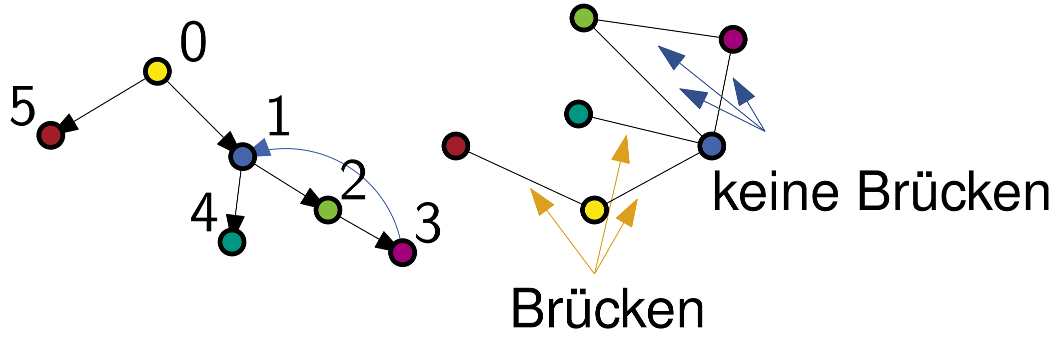
vorher: **lowNode**(v) =„Höchster **Knoten**, der von einem Knoten im Teilbaum von v direkt erreichbar ist“

vorher: **low**(v) =„Höchste **DFS-Nr.**, der von einem Knoten im Teilbaum von v direkt erreichbar ist“

Brücken in ungerichteten Graphen II

jetzt: **low**(v) =„Höchste im Teilbaum von v erreichbare DFS-Nr.“
Beispiel: **low**(●) $= 1$, **low**(●) $= 5$

- Höhe z.B. anhand der DFS-Nummer

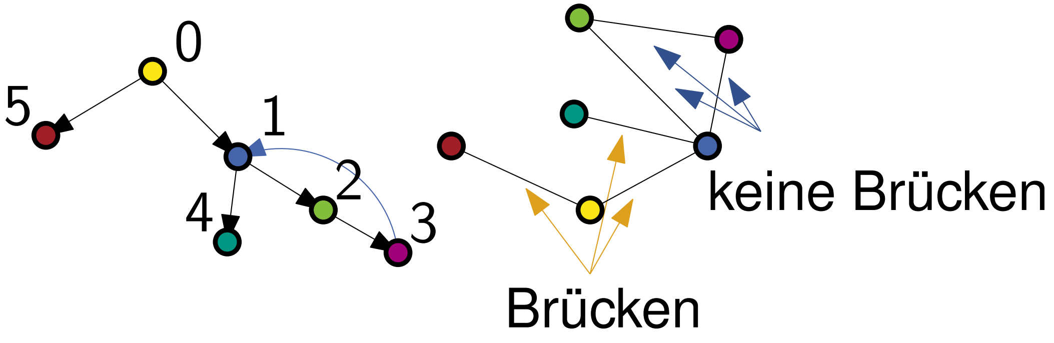


Brücken in ungerichteten Graphen II

jetzt: **low**$(v) =$„Höchste im Teilbaum von v erreichbare DFS-Nr.“
Beispiel: **low**$(\bullet) = 1$, **low**$(\bullet) = 5$

- Höhe z.B. anhand der DFS-Nummer

$\{u, v\}$ ist Brücke $\Leftrightarrow$ $low(v) < dfsNb[u]$



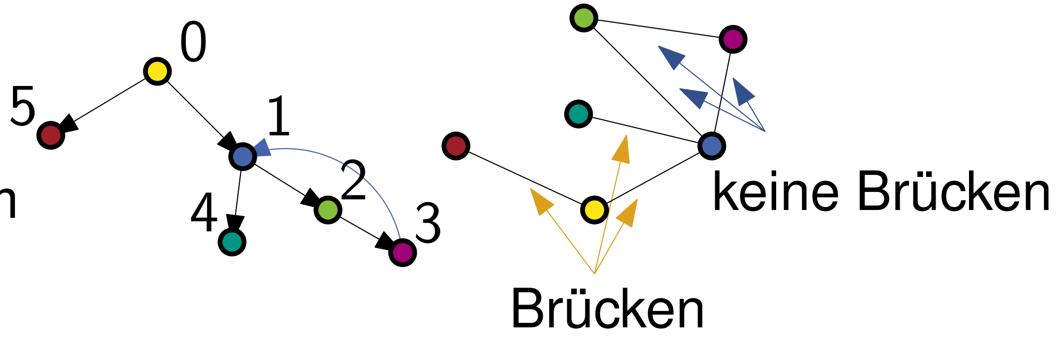
Brücken in ungerichteten Graphen II

jetzt: **low**(v) =„Höchste im Teilbaum von v erreichbare DFS-Nr.“
Beispiel: **low**$(\bullet) = 1$, **low**$(\bullet) = 5$

- Höhe z.B. anhand der DFS-Nummer

$\{u, v\}$ ist Brücke $\Leftrightarrow$ $low(v) < dfsNb[u]$

Frage: Wie berechnet man den **low**-Wert von einem Blatt?

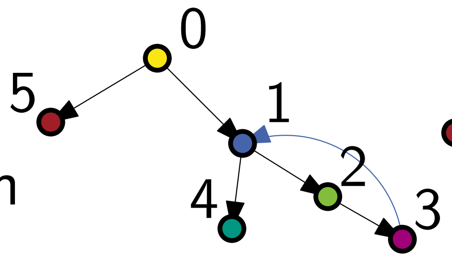


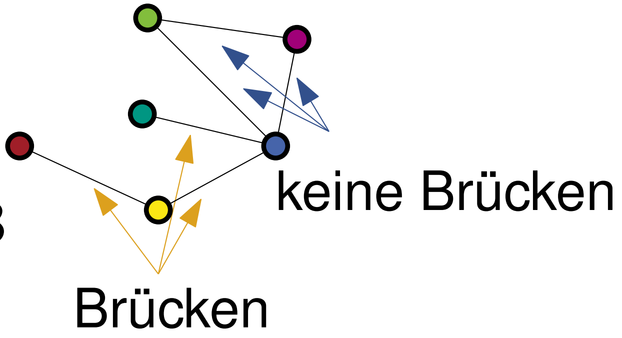
Brücken in ungerichteten Graphen II

jetzt: **low**(v) =„Höchste im Teilbaum von v erreichbare DFS-Nr.“
Beispiel: **low**(●) $= 1$, **low**(●) $= 5$

- Höhe z.B. anhand der DFS-Nummer

$\{u, v\}$ ist Brücke $\Leftrightarrow$ $low(v) < dfsNb[u]$





Frage: Wie berechnet man den **low**-Wert von einem Blatt?

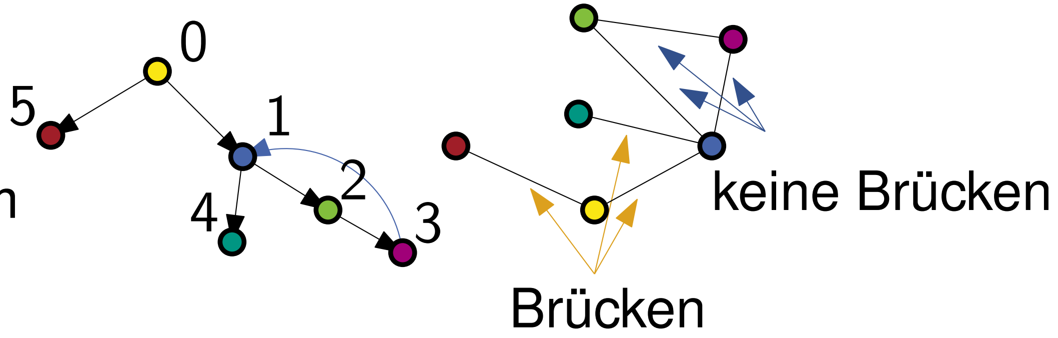
Betrachte ausgehende Kanten im DFS-Baum
Suche Minimum der DFS-Nummern der Zielknoten und eigener DFS-Nr.

Brücken in ungerichteten Graphen II

jetzt: **low**(v) =„Höchste im Teilbaum von v erreichbare DFS-Nr.“
Beispiel: **low**(●) $= 1$, **low**(●) $= 5$

- Höhe z.B. anhand der DFS-Nummer

$\{u, v\}$ ist Brücke $\Leftrightarrow$ $low(v) < dfsNb[u]$



Frage: Wie berechnet man den **low**-Wert von einem Blatt?

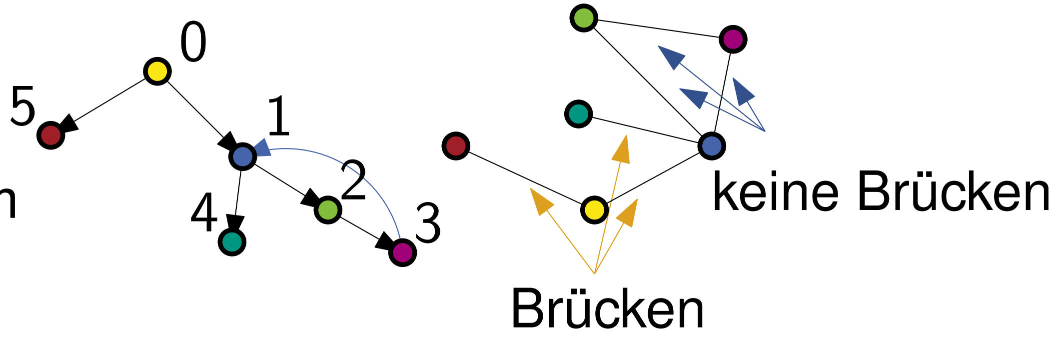
Betrachte ausgehende Kanten im DFS-Baum
Suche Minimum der DFS-Nummern der Zielknoten und eigener DFS-Nr.
(Für innere Knoten zusätzlich noch **low**-Werte der Kindknoten)

Brücken in ungerichteten Graphen II

jetzt: **low**(v) =„Höchste im Teilbaum von v erreichbare DFS-Nr.“
Beispiel: **low**(●) $= 1$, **low**(●) $= 5$

- Höhe z.B. anhand der DFS-Nummer

$\{u, v\}$ ist Brücke $\Leftrightarrow$ $low(v) < dfsNb[u]$



Frage: Wie berechnet man den **low**-Wert von einem Blatt?

Betrachte ausgehende Kanten im DFS-Baum
Suche Minimum der DFS-Nummern der Zielknoten und eigener DFS-Nr.
(Für innere Knoten zusätzlich noch **low**-Werte der Kindknoten)

Geben Sie eine DFS-Variante an, die für jeden Knoten in einem zusammenhängenden, ungerichteten Graphen den **low**-Wert berechnet (5 min)

Setzen des low-Werts über $low[v] \leftarrow \ldots$

Lösung

Geben Sie eine DFS-Variante an, die für jeden Knoten in einem zusammenhängenden, ungerichteten Graphen den **low**-Wert berechnet (5 min)

Frage: Wie berechnet man den **low**-Wert von einem Blatt?
Betrachte ausgehende Kanten im DFS-Baum
Suche Minimum der DFS-Nummern der Zielknoten und eigener DFS-Nr.
(Für innere Knoten zusätzlich noch **low**-Werte der Kindknoten)
Setzen des low-Werts über $low[v] \leftarrow \ldots$

$nextDfsNb \leftarrow 0$

DFS$(u, parent = \perp)$

$dfsNb[u] \leftarrow nextDfsNb{++}$

foreach $v \in N(u)$ **do**

  if $v \notin visited$ **then**

    $visited.insert(v)$

    DFS(v, u)

  else if $dfsNb[v] < dfsNb[u] \wedge parent \neq v$ **then**

  fi

od

Lösung

Geben Sie eine DFS-Variante an, die für jeden Knoten in einem zusammenhängenden, ungerichteten Graphen den **low**-Wert berechnet (5 min)

$nextDfsNb \leftarrow 0$

DFS$(u, parent = \perp)$

$dfsNb[u] \leftarrow nextDfsNb{+}{+}$
$l \leftarrow dfsNb[u]$
foreach $v \in N(u)$ **do**
  if $v \notin visited$ **then**
    $visited.insert(v)$
    DFS(v, u)
    $l \leftarrow \min(l, low[v])$
  else if $dfsNb[v] < dfsNb[u] \wedge parent \neq v$ **then**
    $l \leftarrow \min(l, dfsNb[v])$
  fi
  $low[u] \leftarrow l$
od

Frage: Wie berechnet man den **low**-Wert von einem Blatt?

Betrachte ausgehende Kanten im DFS-Baum

Suche Minimum der DFS-Nummern der Zielknoten und eigener DFS-Nr.

(Für innere Knoten zusätzlich noch **low**-Werte der Kindknoten)

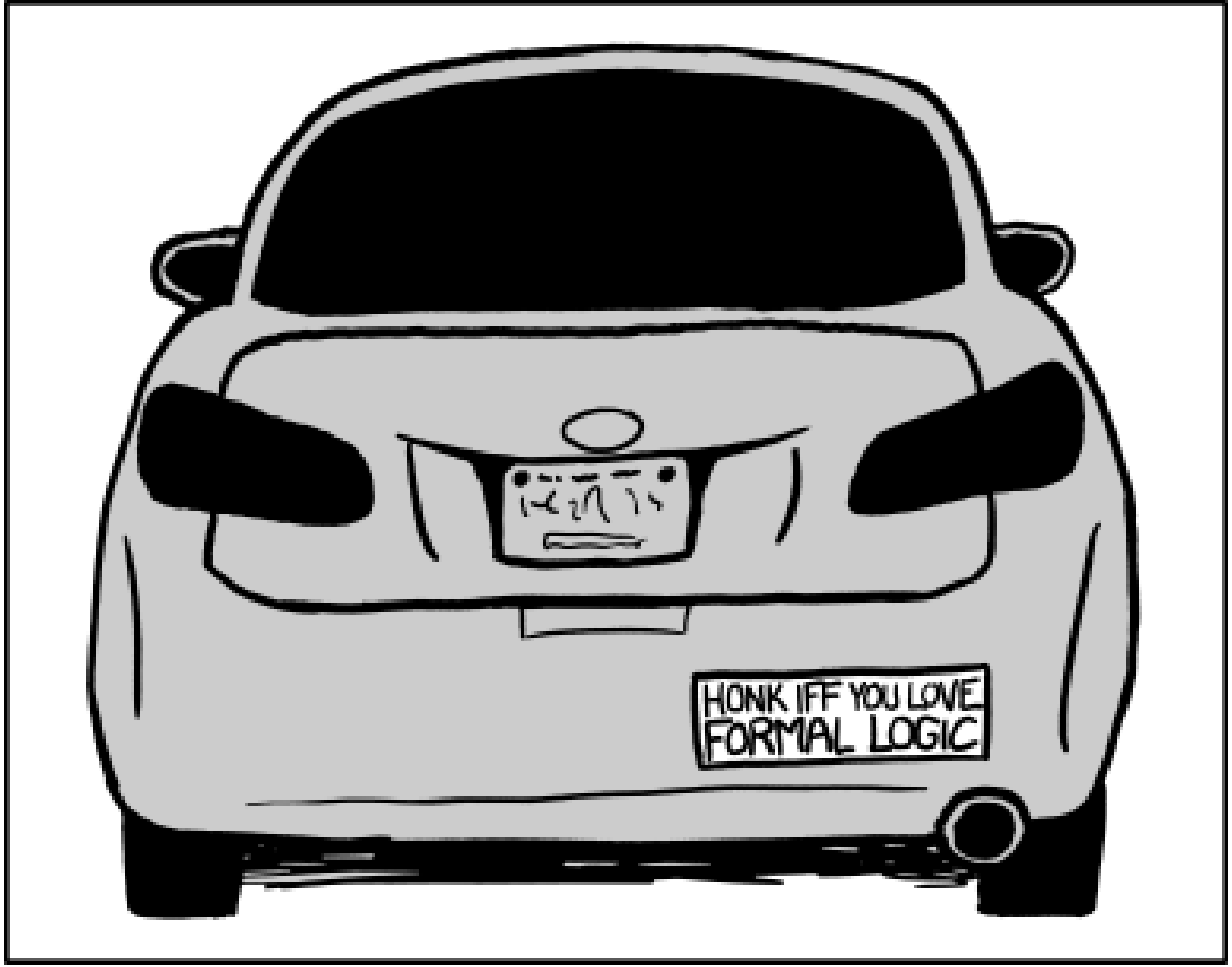
Setzen des low-Werts über $low[v] \leftarrow \ldots$

Peter Bohner - peter.bohner@student.kit.edu

Fragen?

Fragen!

Ende



https://xkcd.com/1033/