Algorithmen und Datenstrukturen 2

Lerneinheit 5: Elementare Graphalgorithmen

Prof. Dr. Christoph Karg

Studiengang Informatik
Hochschule Aalen

Sommersemester 2016

24.5.2016

Einleitung

In dieser Lerneinheit werden elementare Graphalgorithmen durchgenommen. Sie gliedert sich in folgende Abschnitte:

- Datenstrukturen für Graphen
- Breitensuche
- Tiefensuche
- Topologische Sortierung
- Berechnung von Zusammenhangskomponenten

Definition Graph

Ein Graph ist ein Paar $G = (V, E)$, wobei

- V eine endliche Menge von Knoten und
- E eine Menge von Kanten ist.

Man unterscheidet:

- $E \subseteq V \times V$: der Graph ist gerichtet. Ist $(u, v) \in E$, dann ist der Knoten v vom Knoten u aus erreichbar, aber nicht umgekehrt.
- $E \subseteq \{S \subseteq V \mid \|S\| = 2\}$: der Graph ist ungerichtet. Ist $\{u, v\} \in E$, dann ist der Knoten u vom Knoten v aus erreichbar und umkehrt.

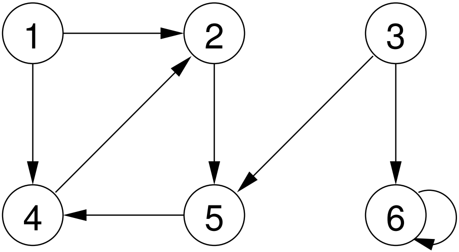
Beachte: Ein ungerichteter Graph ist ein Spezialfall eines gerichteten Graphen mit der Eigenschaft:

$$(u, v) \in E \Longleftrightarrow (v, u) \in E$$

Kürzeste Pfade

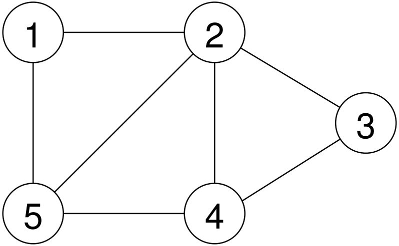
- Ein Pfad von Knoten u nach Knoten v ist eine Folge $\langle w_1, w_2, \ldots, w_n \rangle$ von Knoten wobei
  - ▷ $w_1 = u$, $w_n = v$,
  - ▷ $(w_i, w_{i+1}) \in E$ für alle $i = 1, 2, \ldots, n-1$
- v ist von u aus erreichbar, wenn es einen Pfad von u nach v gibt
- Die Länge eines Pfads ist gleich der Anzahl der Kanten auf dem Pfad
- $\delta(u, v)$ bezeichnet die Länge des kürzesten Pfads von u nach v
- Ist v von u aus erreichbar, dann ist $\delta(u, v)$ die Länge eines kürzesten Pfads von u nach v. Andernfalls ist $\delta(u, v) = \infty$

Beispiel: Gerichteter Graph



- $V = \{1, 2, 3, 4, 5, 6\}$
- $E = \{(1, 2), (1, 4), (2, 5), (4, 2), (3, 5), (3, 6), (5, 4), (6, 6)\}$

Beispiel: Ungerichteter Graph

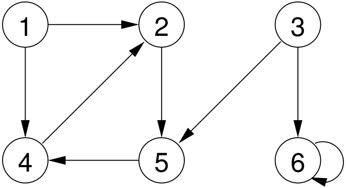


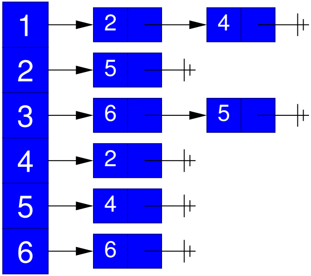
- $V = \{1, 2, 3, 4, 5\}$
- $E = \{\{1, 2\}, \{1, 5\}, \{2, 3\}, \{2, 4\}, \{2, 5\}, \{3, 4\}, \{4, 5\}\}$

Adjazenzlisten

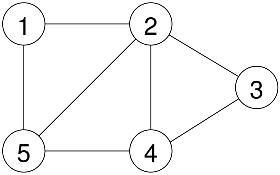
- Eine Adjazenzliste ist eine Datenstruktur zur Speicherung eines Graphen $G = (V, E)$
- Die Struktur besteht aus einem Array Adj der Länge $\|V\|$ von verketteten Listen
- Die Liste $Adj[u]$ enthält alle Knoten v, für die es eine Kante $(u, v) \in E$ gibt
- Die Reihenfolge der Knoten in $Adj[u]$ ist willkürlich
- Vorteil: optimaler Speicherbedarf von $\Theta(\|V\| + \|E\|)$
- Nachteil: Suche der Kanten eines Knotens nicht optimal
- Graphen werden in der Regel mittels Adjazenzlisten gespeichert

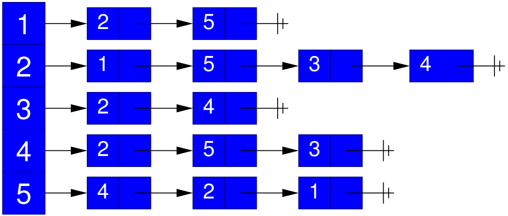
Beispiele Adjazenzliste





Beispiele Adjazenzliste (Forts.)

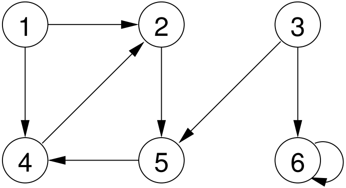




Die Anzahl der Listeneinträge ist $2 \cdot \|E\|$.

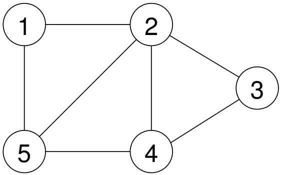
Adjazenzmatrizen

- Eine Adjazenzmatrix ist eine Datenstruktur zur Speicherung eines Graphen $G = (V, E)$
- Die Kanten werden in einer Matrix A der Dimension $\|V\| \times \|V\|$ gespeichert
- Es ist $A[u, v] = 1$ genau dann, wenn eine Kante von Knoten u zu Knoten v führt
- Ist G ein ungerichteter Graph, dann gilt $A^T = A$
- Vorteil: Zugriff auf die Kanteninformation in konstanter Zeit ($\Theta(1)$)
- Nachteil: Speicherplatz quadratisch in $\|V\|$ unabhängig von der Anzahl der Kanten



	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

Beispiele Adjazenzmatrix (Forts.)



	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Breitensuche

- Breitensuche ist einer der einfachsten Algorithmen für die Suche in Graphen
- Der Graph $G = (V, E)$ wird ausgehend von einem Startknoten $s \in V$ durchsucht
- Für jeden von s aus erreichbaren Knoten v werden folgende Informationen berechnet:
  - ▷ Anzahl Kanten eines kürzesten Pfads von s nach v
  - ▷ ein kürzester Pfad von s nach v
- Das Konzept der Breitensuche kommt in vielen anderen Graphalgorithmen zum Einsatz

Idee hinter der Breitensuche

- Idee: Verschiebe die Grenze zwischen entdeckten und unentdeckten Knoten „in der Breite“
- Knoten mit Abstand d zu s werden vor Knoten mit Abstand $d+1$ entdeckt
- Knotenfärbung:
  - ▷ `white` $\rightsquigarrow$ Knoten wurde noch nicht entdeckt
  - ▷ `gray` $\rightsquigarrow$ Knoten wurde entdeckt, aber noch nicht bearbeitet
  - ▷ `black` $\rightsquigarrow$ Knoten wurde entdeckt und bearbeitet
- Weitere Datenstrukturen:
  - ▷ $d[v]$ $\rightsquigarrow$ Abstand des Knotens v zum Knoten s
  - ▷ $\pi[v]$ $\rightsquigarrow$ Vorgänger von v auf dem kürzesten Pfad von s nach v

Algorithmus $\mathrm{BFS}(G, s)$

$\mathrm{BFS}(G, s)$

Input: Graph $G = (V, E)$, Startknoten s

for jeden Knoten $u \in V \setminus \{s\}$ **do**
    $color[u] := \texttt{white}$
    $d[u] := \infty$
    $\pi[u] := \texttt{NIL}$
$color[s] := \texttt{gray}$
$d[s] := 0$
$\pi[s] := \texttt{NIL}$
Initialisiere FIFO Queue Q
$\textsc{Enqueue}(Q, s)$

Algorithmus $\mathrm{BFS}(G, s)$ (Forts.)

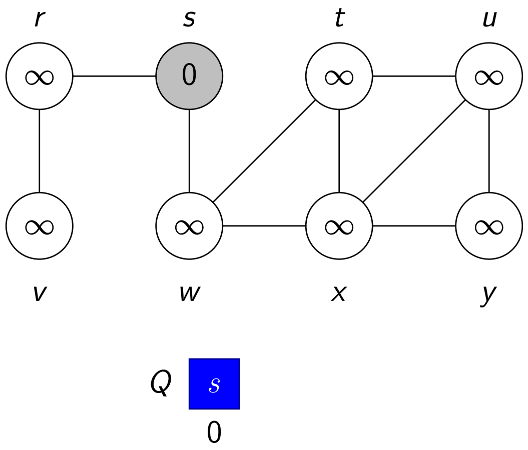
while $Q \neq \emptyset$ **do**
    $u := \textsc{Dequeue}(Q)$
    for jeden Knoten $v \in Adj[u]$ **do**
        if $color[v] = \texttt{white}$ **then**
            $color[v] := \texttt{gray}$
            $d[v] := d[u] + 1$
            $\pi[v] := u$
            $\textsc{Enqueue}(Q, v)$
    $color[u] := \texttt{black}$

Bemerkungen

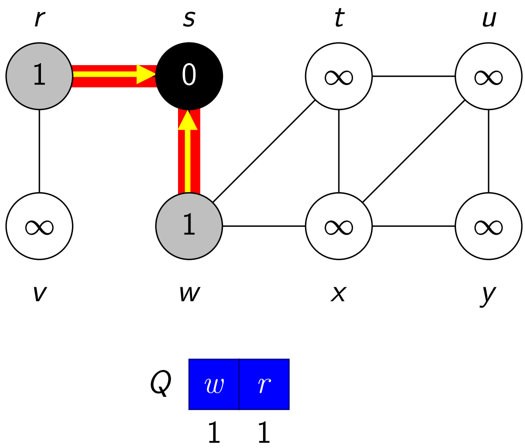
- Die neu entdeckten Knoten werden in der FIFO Warteschlange Q gespeichert
- Es wird jeder Knoten höchstens einmal in Q eingefügt
- Q enthält ausschließlich graue Knoten
- Es werden nur die Knoten entdeckt, die von s aus erreichbar sind
- Die Laufzeit setzt sich zusammen aus:
  - ▷ Initialisierung (Zeilen 1–9): $O(\|V\|)$
  - ▷ Suche (Zeilen 10–18): $O(\|E\|)$

  Insgesamt: $O(\|E\| + \|V\|)$ $\rightsquigarrow$ linear in der Größe der Adjazenzliste

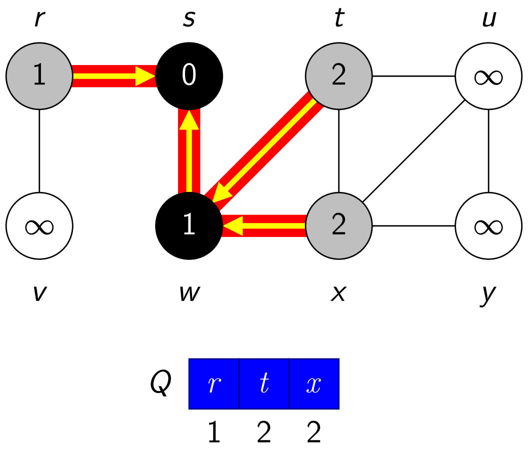
Beispiel Breitensuche



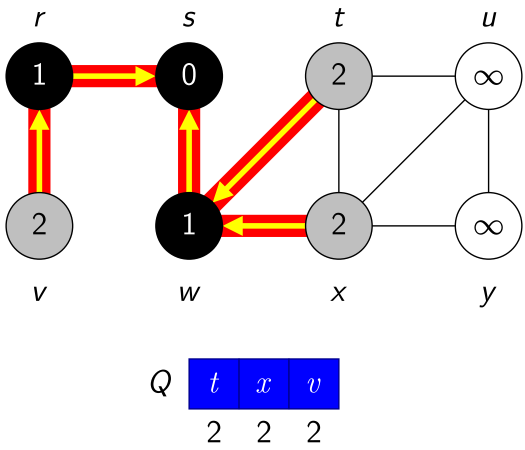
Beispiel Breitensuche (Forts.)



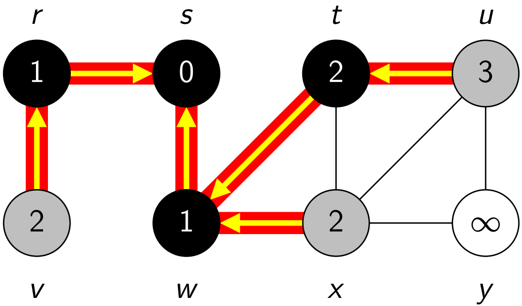
Beispiel Breitensuche (Forts.)

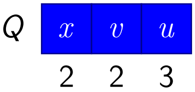


Beispiel Breitensuche (Forts.)

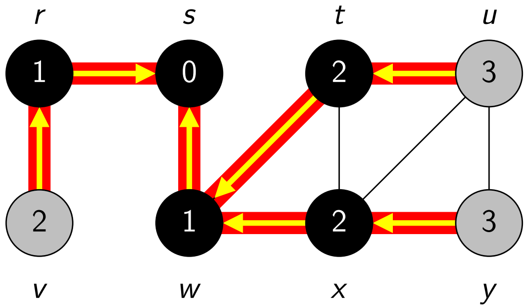


Beispiel Breitensuche (Forts.)





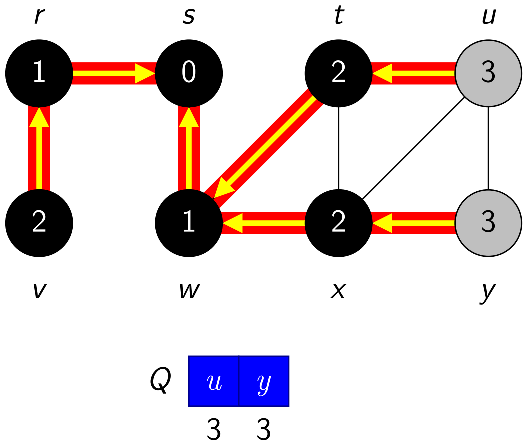
Beispiel Breitensuche (Forts.)



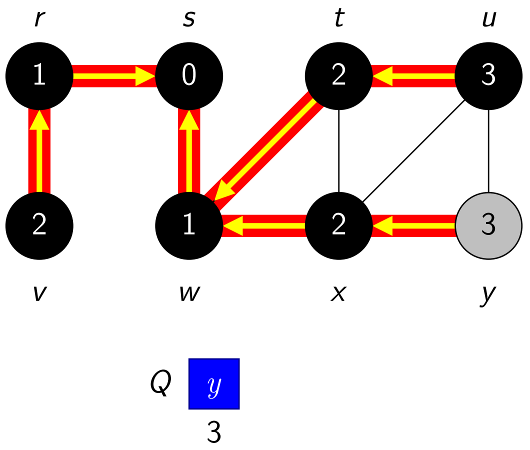
Q

v	u	y
2	3	3

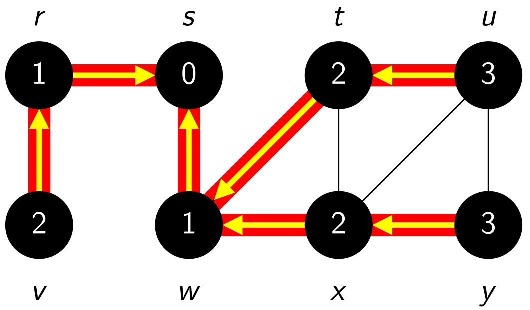
Beispiel Breitensuche (Forts.)



Beispiel Breitensuche (Forts.)



Beispiel Breitensuche (Forts.)



Ergebnis								
n	r	s	t	u	v	w	x	y
$d(n)$	1	0	2	3	2	1	2	3
$\pi(n)$	s	—	w	t	r	s	w	x

Kürzeste Pfade in Graphen

Lemma 1. Sei $G = (V, E)$ ein gerichteter oder ungerichteter Graph. Sei $s \in V$ ein beliebiger Knoten. Für jede Kante $(u, v) \in E$ gilt:

$$\delta(s, v) \leq \delta(s, u) + 1$$

Beweis. Fallunterscheidung:

- Fall 1: u ist von s aus erreichbar. Dann ist auch v von s aus erreichbar. Daher gilt: $\delta(s, v) \leq \delta(s, u) + 1$.
- Fall 2: u ist von s aus nicht erreichbar. Dann ist $\delta(s, u) = \infty$ und die obige Gleichung ist korrekt.

Lemma 2. Sei $G = (V, E)$ ein gerichteter oder ungerichteter Graph. Für jeden Knoten $s \in V$ gilt: Am Ende von $\mathrm{BFS}(G, s)$ ist

$$d[v] \geq \delta(s, v)$$

für alle Knoten $v \in V$.

Beweis. Induktion über die Anzahl der $\textsc{Enqueue}$ Operationen.

Induktionsbehauptung: Für jeden in die Warteschlange eingefügten Knoten v gilt $d[v] \geq \delta(s, v)$.

Induktionsanfang: s ist der erste Knoten, der in die Warteschlange eingefügt wird. Es gilt: $d[s] = 0$ ✓

Breitensuche und kürzeste Pfade (Forts.)

Induktionsschritt: Betrachte einen Knoten v, der während der Verarbeitung des Knotens u entdeckt wird.

- u hat die Warteschlange bereits durchlaufen. Es gilt laut Induktionsbehauptung: $d[u] \geq \delta(s, u)$.
- Wegen Lemma 1 und Zeile 15 von $\mathrm{BFS}(G, s)$ gilt:

$$\begin{aligned} d[v] &= d[u] + 1 \\ &\geq \delta(s, u) + 1 \\ &\geq \delta(s, v) \end{aligned}$$

- v wird in Zeile 17 in die Warteschlange eingefügt. Da er die Farbe `gray` erhält, wird der Wert von $d[v]$ nicht mehr verändert.

Somit ist die Induktionsbehauptung bewiesen.

Breitensuche und kürzeste Pfade (Forts.)

Lemma 3. Betrachte die Warteschlange während der Breitensuche in $G = (V, E)$ mit Startknoten s zu einem beliebigen Zeitpunkt.

Angenommen die Warteschlange enthält die Knoten $v_1, \ldots, v_r$, wobei v_1 und v_r den Anfang bzw. das Ende der Warteschlange bilden.

Dann gilt:

- $d[v_r] \leq d[v_1] + 1$
- $d[v_i] \leq d[v_{i+1}]$ für $i = 1, 2, \ldots, r - 1$

Breitensuche und kürzeste Pfade (Forts.)

Beweis. Induktion über die Anzahl der Operationen der Warteschlange

Induktionsanfang: In Zeile 9 wird s in die Warteschlange eingefügt. Da s der einzige Knoten in der Warteschlange ist, gilt die Behauptung trivialerweise.

Induktionsschritt: Es ist zu zeigen, dass obige Behauptung gilt, nachdem ein Knoten in die Warteschlange eingefügt bzw. aus ihr entfernt wurde.

- Entfernen des Knotens v_1: Dann steht v_2 am Anfang der Liste. Laut Induktionsbehauptung gilt $d[v_1] \leq d[v_2]$. Somit:

  $$d[v_r] \leq d[v_1] + 1 \leq d[v_2] + 1$$

  Die Behauptung ist also korrekt.

Breitensuche und kürzeste Pfade (Forts.)

- Einfügen eines Knotens: Angenommen, der Knoten v wird in Zeile 17 in die Warteschlange eingefügt.

  Dann gibt es einen Knoten u, der zuvor in Zeile 11 aus der Warteschlange entfernt wurde mit folgenden Eigenschaften:

  - ▷ $(u, v) \in E$
  - ▷ $d[v] = d[u] + 1$ (Zeile 15)
  - ▷ $d[u] \leq d[v_1]$ (u war am Anfang der Warteschlange)

  Hieraus folgt:

$$d[v_r] = d[v] = d[u] + 1 \leq d[v_1] + 1$$

  Die Behauptung ist also korrekt.

Breitensuche und kürzeste Pfade (Forts.)

Korollar. Betrachte zwei beliebige Knoten v_i und v_j, die während der Breitensuche in die Warteschlange eingefügt wurden. Angenommen, v_i wurde vor v_j eingefügt. Dann gilt zum Zeitpunkt des Einfügens von v_j die Ungleichung $d[v_i] \leq d[v_j]$.

Beweis. Da der Wert $d[v]$ höchstens einmal während der Breitensuche verändert wird, folgt die Behauptung direkt aus Lemma 3.

Korrektheit der Breitensuche

Satz. Sei G ein gerichteter oder ungerichteter Graph und $s \in V$ ein beliebiger Knoten. Betrachte die Ausführung von $\mathrm{BFS(G,s)}$.

Für jeden Knoten $v \in V$ gilt:

1. $d[v] = \delta(s, v)$
2. Ist v von s aus erreichbar, dann wird v von der Breitensuche entdeckt
3. Ist v von s aus erreichbar, dann ist ein kürzester Pfad von s nach v gleich einem kürzesten Pfad von s nach $\pi[v]$ plus der Kante $(\pi[v], v)$

Korrektheit der Breitensuche (Forts.)

Beweis. Punkt 1: Angenommen, es gibt Knoten v mit $d[v] \neq \delta(s,v)$. Sei v ein solcher Knoten mit minimalen $\delta(s,v)$. Es gilt:

- $v \neq s$, da $d[v] \neq \delta(s,v)$
- $d[v] > \delta(s,v)$ wegen Lemma 2
- v ist von s aus erreichbar (andernfalls wäre $\delta(s,v) = \infty \geq d[v]$)

Betrachte einen kürzesten Pfad von s nach v. Sei u der Knoten, der sich auf diesem Pfad direkt vor v befindet. Also gilt $\delta(s,v) = \delta(s,u) + 1$.

Wegen $\delta(s,u) < \delta(s,v)$ und der Wahl von v muss $d[u] = \delta(s,u)$ gelten. Insgesamt:

$$d[v] > \delta(s,v) = \delta(s,u) + 1 = d[u] + 1 \tag{1}$$

Korrektheit der Breitensuche (Forts.)

Betrachte nun den Zeitpunkt, zu dem u aus der Warteschlange entfernt wird.

- Fall 1: $\mathit{color}[v] = \texttt{white}$: Nach Ausführung von Zeile 15 gilt $d[v] = d[u] + 1$. Widerspruch zu Ungleichung (1)
- Fall 2: $\mathit{color}[v] = \texttt{black}$: Dann wurde v bereits aus der Warteschlange entfernt. Wegen obigem Korollar gilt: $d[v] \leq d[u]$. Widerspruch zu Ungleichung (1)
- Fall 3: $\mathit{color}[v] = \texttt{gray}$: Dann wurde v grau gefärbt bei der Verarbeitung eines Knotens w, der sich vor u in der Warteschlange befand. Wegen dem obigem Korollar folgt

  $$d[v] = d[w] + 1 \leq d[u] + 1$$

  Widerspruch zu Ungleichung (1)

Korrektheit der Breitensuche (Forts.)

Somit gilt $d[v] = \delta(s, v)$ für alle $v \in V$.

Punkt 2: folgt direkt aus Punkt 1. Angenommen ein Knoten v ist von s aus erreichbar, wird aber von der Breitensuche nicht gefunden. Dann ist $d[v] = \infty \neq \delta(s, v)$. Widerspruch!

Punkt 3: Aus $\pi[v] = u$ folgt, dass $d[v] = d[u] + 1$ ist. Also erhält man einen kürzesten Pfad von s nach v, indem man einen kürzesten Pfad von s nach $u = \pi[v]$ um die Kante $(u, v) = (\pi[v], v))$ erweitert.

Breadth-First Bäume

- Der Vorgängergraph einer Breitensuche in G mit Startknoten s ist $G_\pi = (V_\pi, E_\pi)$
- G_π wird anhand der Tabelle π definiert:
  - ▷ $V_\pi = \{v \in V \mid \pi(v) \neq \mathtt{NIL}\} \cup \{s\}$
  - ▷ $E_\pi = \{(\pi[v], v) \mid v \in V_\pi \setminus \{s\}\}$
- G_π enthält alle Knoten, die von s aus erreichbar sind.
- G_π ist ein Baum mit Wurzel s. In Tiefe d befinden sich alle Knoten mit Abstand d zu s.
- Der Graph G_π wird Breadth-First Baum genannt.

Ausgabe eines kürzesten Pfads

```
PRINTPATH(G, s, v)
if v = s then
    print „s“
else
    if π[v] = NIL then
        print „Kein Pfad von s nach v“
    else
        PRINTPATH(G, s, π[v])
        print „v“
```

Laufzeit: $O(\|V\|)$

Tiefensuche

- Strategie: Suche möglichst „tief“ im Graph
- Es kommt die Backtracking Programmiertechnik zum Einsatz
- Rekursion: Falls von u eine Kante zu einem unentdeckten Knoten v führt, dann durchsuche v. Ansonsten setze die Bearbeitung beim Vorgänger von u fort
- Knotenfärbung
  - ▷ `white` $\rightsquigarrow$ Knoten wurde noch nicht entdeckt
  - ▷ `gray` $\rightsquigarrow$ Knoten wurde entdeckt, aber noch nicht komplett bearbeitet
  - ▷ `black` $\rightsquigarrow$ Knoten wurde entdeckt und komplett bearbeitet

Tiefensuche (Forts.)

- Jeder Knoten u erhält zwei Zeitstempel:
  - ▷ $d[u] \rightsquigarrow$ Zeitpunkt der Entdeckung von u
  - ▷ $f[u] \rightsquigarrow$ Zeitpunkt der vollständigen Bearbeitung von u
- Für jeden Knoten u gilt: $1 \leq d[u] < f[u] \leq 2 \cdot \|V\|$
- Für die Färbung von u zum Zeitpunkt t gilt:
  - ▷ $t < d[u] \rightsquigarrow \mathit{color}[u] = \texttt{white}$
  - ▷ $d[u] \leq t < f[u] \rightsquigarrow \mathit{color}[u] = \texttt{gray}$
  - ▷ $f[u] \leq t \rightsquigarrow \mathit{color}[u] = \texttt{black}$

Algorithmus $\mathrm{DFS}(G)$

$\mathrm{DFS}(G)$

Input: Graph $G = (V, E)$

for jeden Knoten $u \in V$ **do**
    $color[u] := \texttt{white}$
    $\pi[u] := \texttt{NIL}$
$time := 0$
for jeden Knoten $u \in V$ **do**
    if $color[u] = \texttt{white}$ **then**
        $\mathrm{DFSV{\scriptsize ISIT}}(u)$

Algorithmus $\textsc{DFSVisit}(u)$

$\textsc{DFSVisit}(u)$

$\mathit{color}[u] := \texttt{gray}$
$\mathit{time} := \mathit{time} + 1$
$d[u] := \mathit{time}$
for jeden Knoten $v \in \mathit{Adj}[u]$ **do**
    if $\mathit{color}[v] = \texttt{white}$ **then**
        $\pi[v] := u$
        $\textsc{DFSVisit}(v)$
$\mathit{color}[u] := \texttt{black}$
$\mathit{time} := \mathit{time} + 1$
$f[u] := \mathit{time}$

Bemerkungen

- $\text{DFSVISIT}(u)$ sucht mittels Rekursion tiefstmöglich im Baum
- Die Laufzeit von $\text{DFS}(G)$ ohne Berücksichtigung der DFSVISIT Aufrufe ist $\Theta(\|V\|)$
- Für jeden Knoten u wird genau einmal $\text{DFSVISIT}(u)$ aufgerufen. Die Laufzeit aller DFSVISIT Aufrufe ist daher $\Theta(\|E\|)$
- Die Laufzeit von $\text{DFS}(G)$ ist $\Theta(\|V\| + \|E\|)$
- Das Ergebnis von $\text{DFS}(G)$ ist eine Menge von Bäumen, der sog. Depth-First Wald

Depth-First Wald

- Der Vorgängergraph einer Tiefensuche in G ist der Graph $G_\pi = (V, E_\pi)$, wobei

$$E_\pi = \{(\pi[v], v) \mid v \in V \text{ und } \pi[v] \neq \texttt{NIL}\}$$

- Die Kanten in E_π nennt man Baumkanten
- G_π ist ein Wald bestehend aus einem oder mehreren Depth-First Bäumen
- G_π nennt man Depth-First Wald

Klassifikation der Kanten

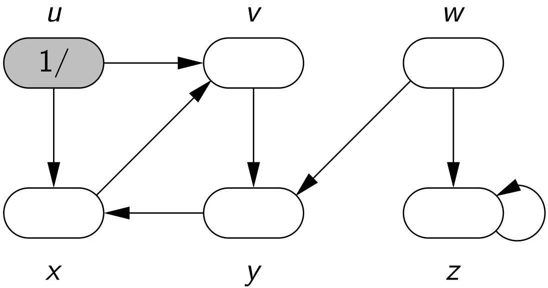
Anhand der Tiefensuche kann man jede Kante $(u, v) \in E$ wie folgt einteilen:

- Tree Edge: $(u, v) \in E_\pi$, d.h., v wurde über die Kante (u, v) entdeckt
- Back Edge: die Kante verbindet u mit einem seiner Vorfahren v
- Forward Edge: die Kante verbindet u mit einem seiner Nachkommen v
- Cross Edge: jede andere Kante

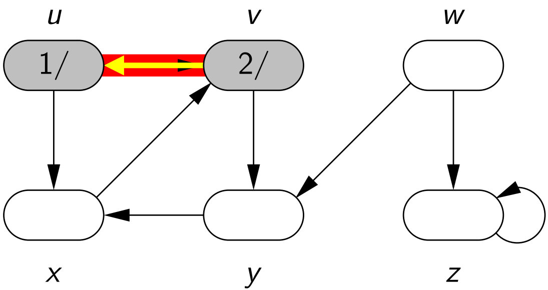
Klassifikation der Kanten (Forts.)

- Mittels modifizierter Tiefensuche kann die Kantenklassifikation berechnet werden
- Wird die Kante (u, v) zum ersten Mal benutzt, dann ist sie eine
  - Tree Edge, falls $color[v] = \texttt{white}$
  - Back Edge, falls $color[v] = \texttt{gray}$
  - Forward/Cross Edge, falls $color[v] = \texttt{black}$
- Die Unterscheidung zwischen Forward und Cross Edges ist erst nach der Tiefensuche möglich

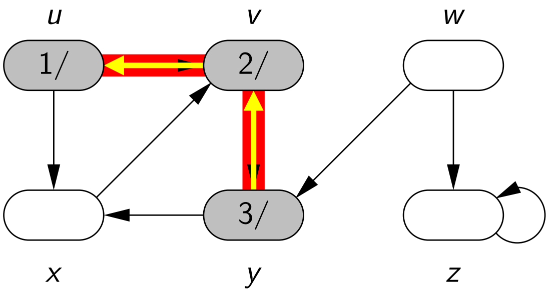
Beispiel Tiefensuche



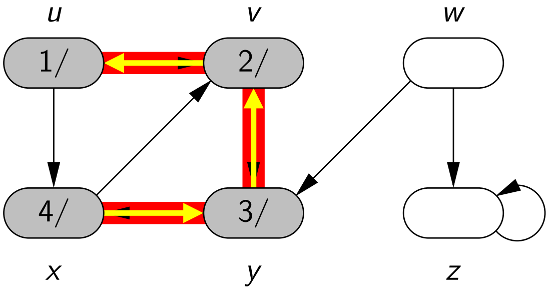
Beispiel Tiefensuche (Forts.)



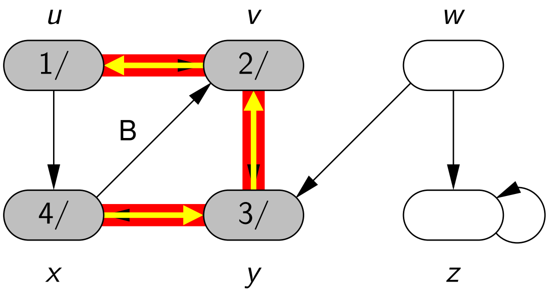
Beispiel Tiefensuche (Forts.)



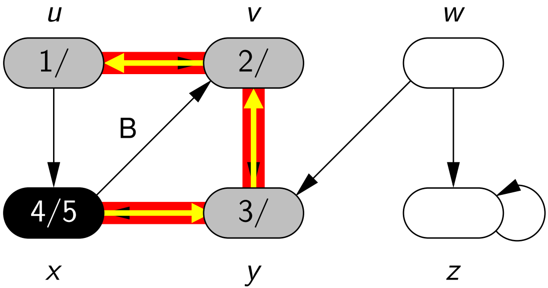
Beispiel Tiefensuche (Forts.)



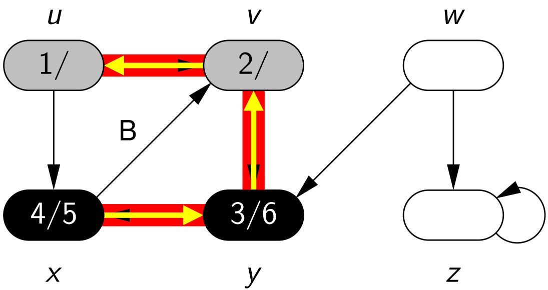
Beispiel Tiefensuche (Forts.)



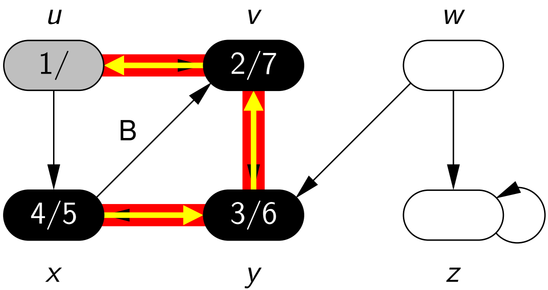
Beispiel Tiefensuche (Forts.)



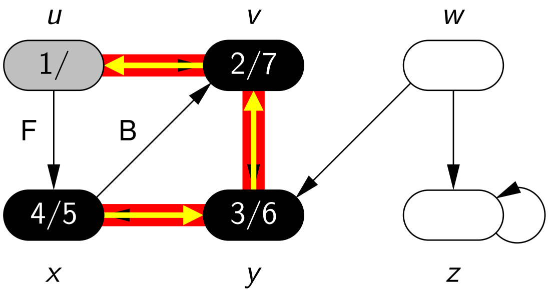
Beispiel Tiefensuche (Forts.)



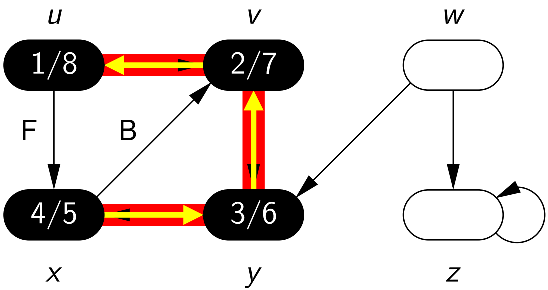
Beispiel Tiefensuche (Forts.)



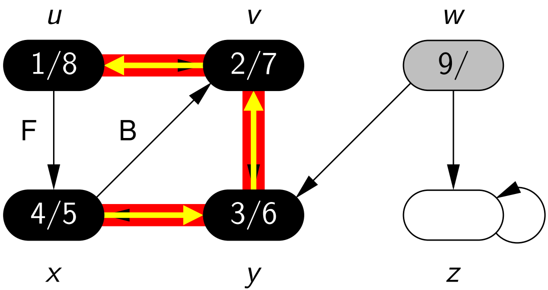
Beispiel Tiefensuche (Forts.)



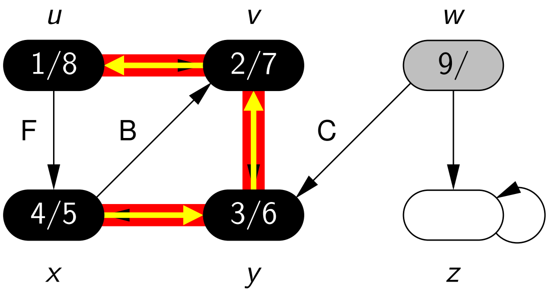
Beispiel Tiefensuche (Forts.)



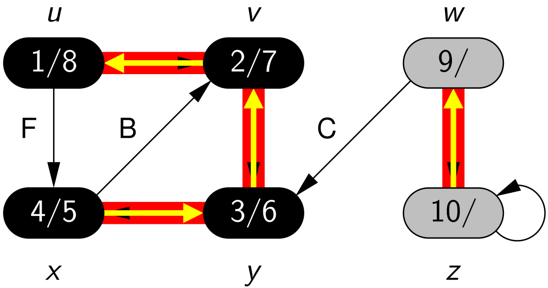
Beispiel Tiefensuche (Forts.)



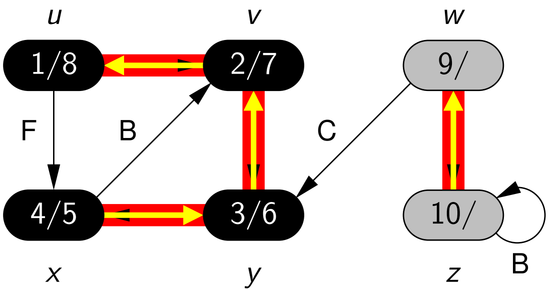
Beispiel Tiefensuche (Forts.)



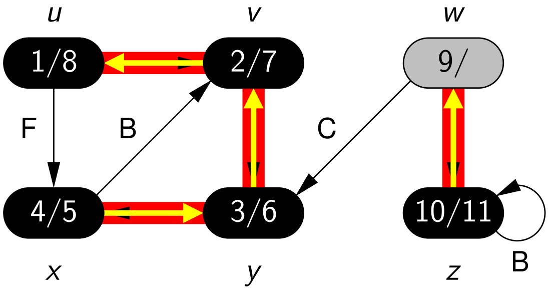
Beispiel Tiefensuche (Forts.)



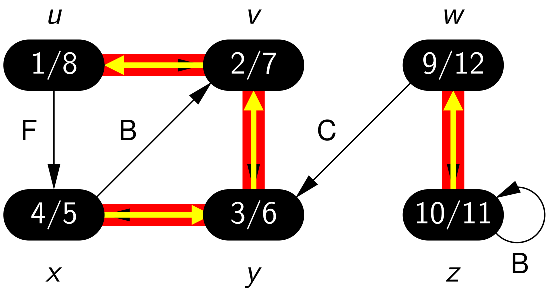
Beispiel Tiefensuche (Forts.)



Beispiel Tiefensuche (Forts.)



Beispiel Tiefensuche (Forts.)



Klammerungssatz

Klammerungssatz. Sei $G = (V, E)$ ein (gerichteter oder ungerichteter) Graph. Nach der Tiefensuche in G gilt für jedes Paar von Knoten u und v genau eine der folgenden Bedingungen:

- die Intervalle $[d[u], f[u]]$ und $[d[v], f[v]]$ sind disjunkt und weder u noch v sind ein Nachkomme des anderen Knotens im Depth-First Wald
- das Intervall $[d[u], f[u]]$ ist vollständig im Intervall $[d[v], f[v]]$ enthalten und u ist ein Nachkomme von v in einem Depth-First Baum
- das Intervall $[d[v], f[v]]$ ist vollständig im Intervall $[d[u], f[u]]$ enthalten und v ist ein Nachkomme von u in einem Depth-First Baum

Klammerungssatz (Forts.)

Beweis. Fall 1: $d[u] < d[v]$. Angenommen, $d[v] < f[u]$.

- Dann wurde v entdeckt, während u grau gefärbt war. v ist also ein Nachkomme von u.
- Wegen der Rekursion wird der Knoten v komplett bearbeitet, bevor die Bearbeitung von u abgeschlossen wird. Somit: $f[v] < f[u]$
- Ergebnis: das Intervall $[d[v], f[v]]$ ist im Intervall $[d[u], f[u]]$ enthalten.

Klammerungssatz (Forts.)

Angenommen, $d[v] > f[u]$.

- Bekanntlich gilt für jeden Knoten w, dass $d[w] < f[w]$.
- Also: $d[u] < f[u] < d[v] < f[v]$
- Konsequenz: die Intervalle $[d[u], f[u]]$ und $[d[v], f[v]]$ sind disjunkt.

Fall 2: $d[u] > d[v]$. Analog zu Fall 1.

Verschachtelungssatz

Verschachtelungssatz. Der Knoten v ist ein echter Nachkomme des Knotens u in einem Depth-First Baum für einen (gerichteten oder ungerichteten) Graphen G genau dann, wenn $d[u] < d[v] < f[v] < f[u]$.

Beweis. folgt direkt aus dem Klammerungssatz!

Weißer-Pfad-Satz

Weißer-Pfad-Satz. In einer Tiefensuche eines (gerichteten oder ungerichteten) Graphen G ist der Knoten v genau dann ein Nachkomme des Knotens u, wenn zum Zeitpunkt $d[u]$ v von u aus erreichbar ist über einen Pfad, der ausschließlich weiße Knoten enthält.

Beweis. "$\Rightarrow$": Angenommen, v ist ein Nachkomme von u.

Sei w ein beliebiger Knoten auf einem Pfad von u nach v, so dass w ein echter Nachkomme von u ist.

Wegen des Verschachtelungssatzes muss $d[u] < d[w]$ gelten.

Also muss w zum Zeitpunkt $d[u]$ weiß gefärbt sein.

Weißer-Pfad-Satz (Forts.)

"$\Leftarrow$": Angenommen, zum Zeitpunkt $d[u]$ ist v von u aus erreichbar über einen Pfad, der ausschließlich weiße Knoten enthält, aber v wird kein Nachkomme von u in dem Depth-First Baum.

Der Einfachheit halber sei angenommen, dass alle Knoten vor v Nachkommen von u in dem Depth-First Baum sind.

Sei w der Knoten auf dem Pfad, der unmittelbar vor v liegt. Es gilt wegen des Verschachtelungssatzes, dass $f[w] \leq f[u]$.

Weißer-Pfad-Satz (Forts.)

Da v von w aus erreichbar ist, wird v entdeckt bevor die Bearbeitung von w abgeschlossen ist. Also gilt:

$$d[u] < d[w] < d[v] < f[v] < f[w] \leq f[u]$$

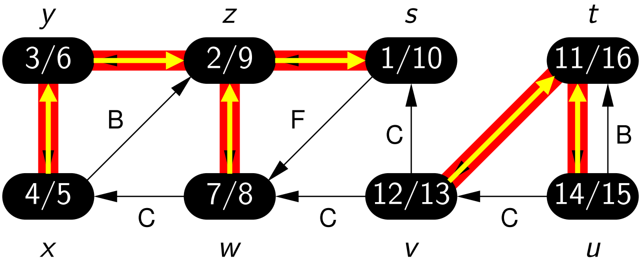
Mit dem Klammerungssatz folgt, dass das Intervall $[d[v], f[v]]$ komplett im Intervall $[d[u], f[u]]$ enthalten sein muss.

Mit dem Verschachtelungssatz folgt, dass v ein Nachkomme von u sein muss.

Widerspruch! Also ist obige Annahme falsch.

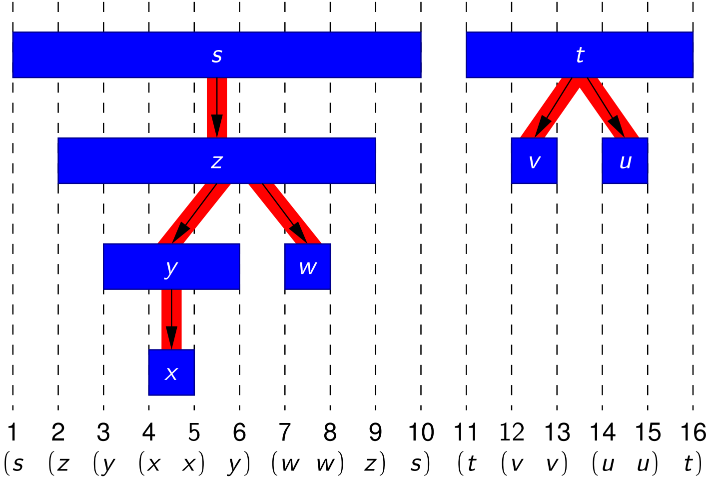
Beispiel Eigenschaften Tiefensuche

Ergebnis einer Tiefensuche:



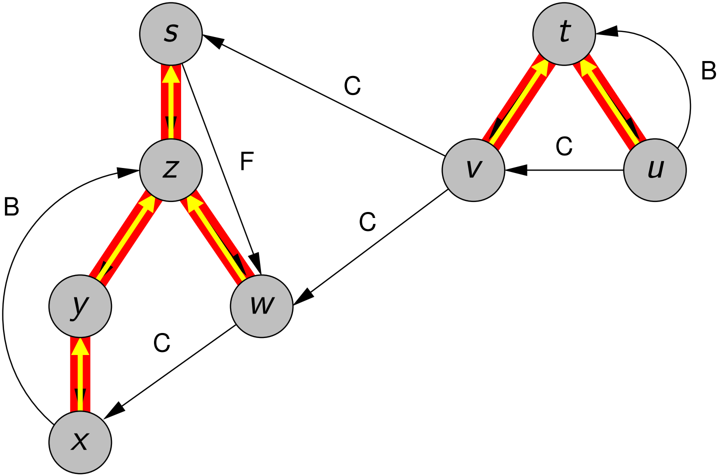
Beispiel Eigenschaften Tiefensuche (Forts.)

Die Tiefensuche erzeugt eine Klammerungsstruktur:



Beispiel Eigenschaften Tiefensuche (Forts.)

Der Depth-First Wald mit den verschiedenen Kantentypen:



Topologische Sortierung

Die topologische Sortierung eines gerichteten azyklischen Graphen ist eine horizontale Anordnung der Knoten so dass die Kanten ausschließlich von links nach rechts zeigen.

Formal: es gibt eine Knotenordnung

$$ord : V \mapsto \{1, \ldots, \|V\|\}$$

so dass für alle Kanten $(u, v) \in E$ die Ungleichung

$$ord(u) < ord(v)$$

gilt.

Beachte: Enthält der Graph einen Zyklus, dann kann man ihn nicht topologisch sortieren.

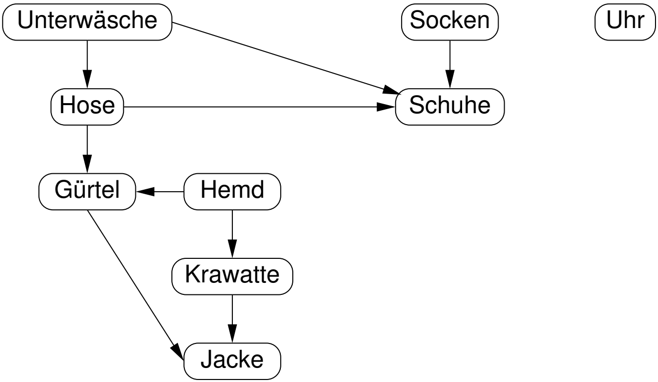
Algorithmus TopologicalSort(G)

TopologicalSort(G)

```
Berechne mittels Tiefensuche für alle Knoten u
die Abschlusszeiten f[u]
if eine Back Edge gefunden then
   print „Graph enthält einen Zyklus“
else
   Wenn die Bearbeitung eines Knotens beendet ist,
   dann füge ihn am Anfang der verketteten Liste ein
return Zeiger auf den Anfang der verketteten Liste
```

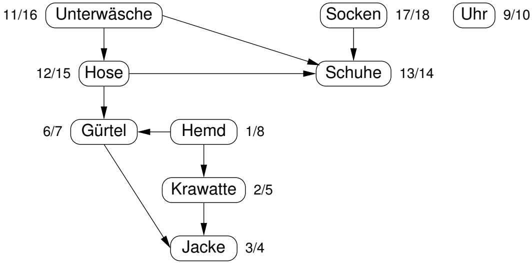
Laufzeit: $\Theta(\|V\| + \|E\|)$

Topologische Sortierung Beispiel

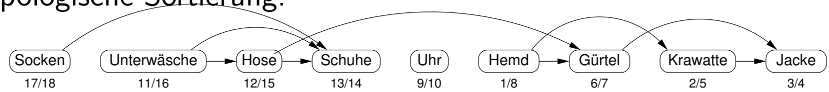


Topologische Sortierung Beispiel (Forts.)

Ergebnis der Tiefensuche:



Topologische Sortierung:



Ein nützliches Lemma

Lemma. Ein gerichteter Graph G ist genau dann azyklisch, wenn die Tiefensuche in G keine Back Edges findet.

Beweis. "$\Rightarrow$": Angenommen G ist azyklisch und die Tiefensuche findet eine Back Edge (u, v). Dann ist v ein Vorfahre von u im Tiefensuche-Wald. Demnach gibt es in G einen Pfad von v nach u. Die Kante (u, v) vervollständigt den Zyklus. Widerspruch!

"$\Leftarrow$": Angenommen, in G gibt es einen Zyklus c. Sei v der Knoten in c, der von der Tiefensuche als erstes gefunden wird und sei (u, v) die zugehörige Kante in c.

Zum Zeitpunkt $d[v]$ bilden die Knoten in c einen weißen Pfad von v nach u. Weißer-Pfad-Satz: u ist ein Nachkomme von v.

Die Kante (u, v) ist also eine Back-Edge.

Korrektheit von $\textsc{TopologicalSort}(G)$

Satz. $\textsc{TopologicalSort}(G)$ berechnet eine topologische Sortierung eines gerichteten azyklischen Graphen.

Beweis. Angenommen, die Endzeiten der Knoten eines gerichteten Graphen $G = (V, E)$ werden durch eine Tiefensuche ermittelt.

Behauptung: Für jedes Paar von Knoten $u, v \in V$ gilt: Falls es in G eine Kante von u nach v gibt, dann ist $f[v] < f[u]$.

Betrachte eine beliebige Kante $(u, v) \in E$, die von der Tiefensuche untersucht wird.

Wenn die Kante (u, v) untersucht wird, dann kann v nicht grau sein. Andernfalls wäre v ein Vorfahre von u und (u, v) somit eine Back-Edge. Dies steht im Widerspruch zu obigem Lemma.

Korrektheit (Forts.)

Fallunterscheidung:

- Fall 1: v ist weiß. Dann wird v ein Nachkomme von u, d.h., $f[v] < f[u]$.
- Fall 2: v ist schwarz. Dann ist die Bearbeitung von v bereits beendet und der Wert von $f[v]$ bereits festgelegt. Da u noch bearbeitet wird, folgt, dass $f[v] < f[u]$.

Sei $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_{n-1} \rightarrow v_n$ die von $\textsc{TopologicalSort}(G)$ berechnete Knotenliste. Es gilt:

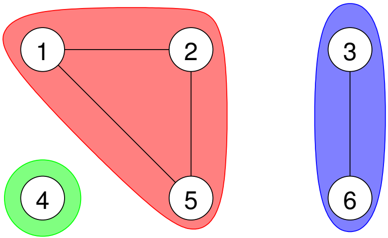
$$f[v_1] > f[v_2] > \ldots > f[v_n]$$

Wegen obiger Behauptung folgt, dass diese Anordnung eine topologische Sortierung von G ist.

Zusammenhangskomponenten

- Ein ungerichteter Graph ist zusammenhängend, wenn jedes Knotenpaar durch einen Pfad verbunden ist
- Die Zusammenhangskomponenten eines ungerichteten Graphen sind die Äquivalenzklassen der “ist erreichbar von” Relation über der Knotenmenge

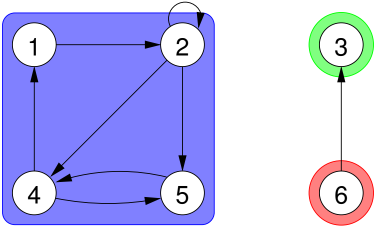
Beispiel:



Zusammenhangskomponenten (Forts.)

- Ein gerichteter Graph ist stark zusammenhängend, wenn jeder Knoten von jedem anderen Knoten aus erreichbar ist
- Die starken Zusammenhangskomponenten eines gerichteten Graphen sind die Äquivalenzklassen der “sind gegenseitig erreichbar” Relation über der Knotenmenge

Beispiel:



Transponieren eines Graphen G

Sei $G = (V, E)$ ein gerichteter Graph. Der transponierte Graph von G ist $G^T = (V, E^T)$, wobei

$$E^T = \{(v, u) \mid (u, v) \in E\}.$$

Beachte:

- Gibt es in G einen Pfad von u nach v, dann gibt es in G^T einen Pfad von v nach u
- G und G^T haben dieselben starken Zusammenhangskomponenten

Algorithmus

STRONGLYCONNECTEDCOMPONENTS(G)

Input: Gerichteter Graph $G = (V, E)$

Output: Starke Zusammenhangskomponenten von G

1 *Berechne mittels einer Tiefensuche in G die Endzeiten $f[u]$ für alle Knoten $u \in V$*

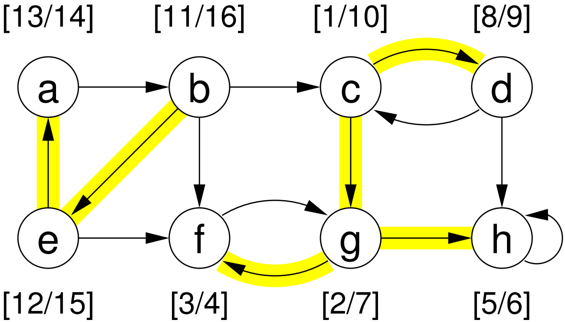
2 *Berechne G^T*

3 *Führe in G^T eine Tiefensuche durch, wobei in Zeile 5 von* $\mathrm{DFS}(G^T)$ *die Knoten mit absteigendem $f[u]$ (von Zeile 1) ausgewählt werden*

4 *Gebe die Knoten jedes Baums des in Zeile 3 berechneten Depth-First Walds als Zusammenhangkomponente aus*

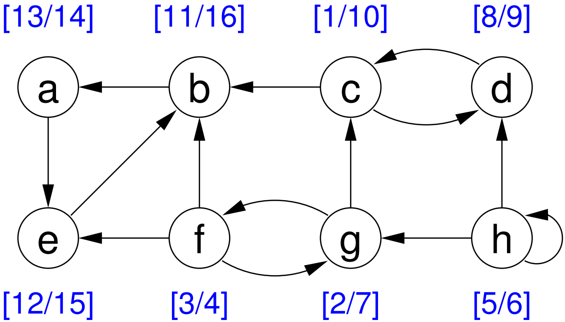
Laufzeit: $\Theta(\|V\| + \|E\|)$

Beispiel



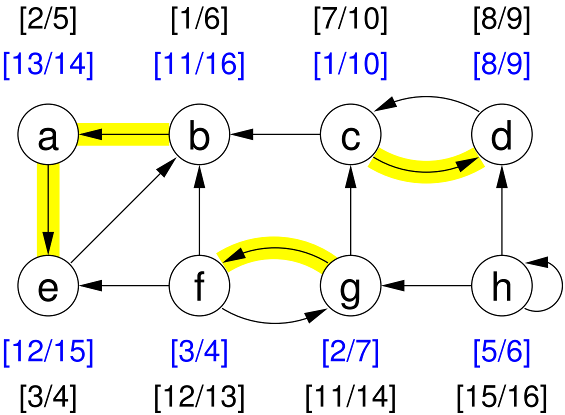
Ergebnis der ersten Tiefensuche

Beispiel (Forts.)



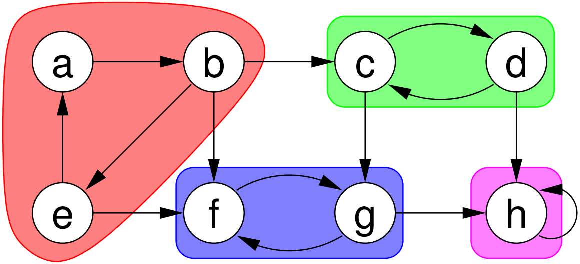
Berechnung von G^T

Beispiel (Forts.)



Ergebnis der Tiefensuche von G^T
(Reihenfolge der Knoten gemäß $f[u]$ der ersten Tiefensuche)

Beispiel (Forts.)



Ergebnis

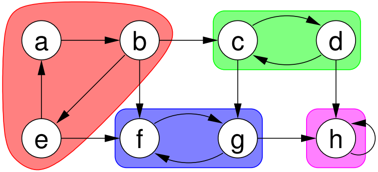
Komponentengraph von G

Sei $G = (V, E)$ ein gerichteter Graph. Seien $C_1, \ldots, C_k$ die starken Zusammenhangskomponenten von G.

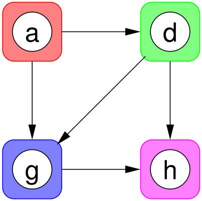
Der Komponentengraph $G^{SCC} = (V^{SCC}, E^{SCC})$ ist wie folgt definiert:

- $V^{SCC} = \{v_1, \ldots, v_k\}$, wobei $v_i \in C_i$ für $i = 1, \ldots, k$
- $E^{SCC} = \{(v_i, v_j) \mid \exists (u, v) \in E : u \in C_i \text{ und } v \in C_j\}$

Beispiel (Forts.)



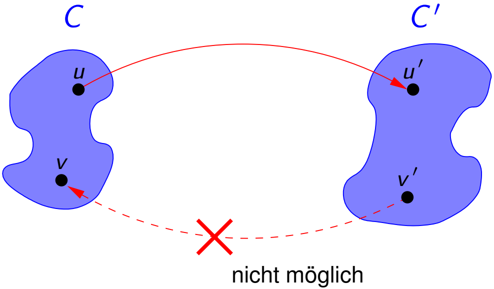
Graph G und ...



...der zugehörige Komponentengraph G^{SCC}

Korrektheit

Lemma 1. Seien C und C' zwei verschiedene starke Zusammenhangskomponenten des Graphen $G = (V, E)$. Seien $u, v \in C$ und $u', v' \in C'$. Angenommen, Es gibt einen Pfad $u \rightsquigarrow u'$ in G. Dann gibt es keinen Pfad $v' \rightsquigarrow v$ in G.



Korrektheit (Forts.)

Beweis. Angenommen, es gibt einen Pfad $v' \rightsquigarrow v$ in G. Dann gibt es in G die Pfade $u \rightsquigarrow v \rightsquigarrow v'$ und $v' \rightsquigarrow v \rightsquigarrow u$.

Also sind u und v' gegenseitig erreichbar. Dies ist ein Widerspruch, da C und C' disjunkt sind.

Korollar. G^{SCC} ist ein gerichteter, azyklischer Graph.

Korrektheit (Forts.)

Vereinbarung: im folgenden werden ausschließlich die Start- und Endzeiten der ersten Tiefensuche betrachtet.

Notation: Für eine Knotenmenge $U \subseteq V$ definiere:

- $d[U] = \min\{d[u] \mid u \in U\}$
- $f[U] = \max\{f[u] \mid u \in U\}$

Lemma 2. Seien C und C' verschiedene starke Zusammenhangskomponenten in einem Graphen G. Angenommen, es gibt eine Kante (u, v), wobei $u \in C$ und $v \in C'$. Dann gilt $f(C) > f(C')$.

Korrektheit (Forts.)

Beweis. Fall 1: $d[C] < d[C']$. Sei x der erste Knoten in C, der während der (ersten) Tiefensuche entdeckt wird.

Zum Zeitpunkt $d[x]$ gilt:

- Alle Knoten in C und C' sind weiß
- Von x führt zu jedem Knoten in C ein Pfad mit ausschließlich weißen Knoten
- Da $(u, v) \in E$, führt von x zu jedem Knoten $w \in C'$ ein Pfad mit ausschließlich weißen Knoten. Der Pfad ist

$$x \rightsquigarrow u \longrightarrow v \rightsquigarrow w.$$

Korrektheit (Forts.)

Weißer-Pfad-Satz: Alle Knoten in C und C' sind Nachkommen von x in dem von der Tiefensuche berechneten Depth-First-Baum.

Verschachtelungssatz: $f(x) = f(C) > f(C')$

Fall 2: $d[C] > d[C']$. Sei y der erste Knoten in C', der während der (ersten) Tiefensuche entdeckt wird.

Zum Zeitpunkt $d[y]$ gilt:

- Alle Knoten in C' sind weiß
- Es führt von y zu jedem Knoten in C' ein Pfad mit ausschließlich weißen Knoten

Korrektheit (Forts.)

Weißer-Pfad-Satz: Alle Knoten in C' sind Nachfolger von y im Depth-First-Baum.

Verschachtelungssatz: $f[y] = f(C')$.

Zum Zeitpunkt $d[y]$ sind alle Knoten in C weiß. Da es eine Kante (u, v) von C nach C' gibt, folgt mit Lemma 1, dass kein Pfad von C' nach C existiert.

Also sind zum Zeitpunkt $f[y]$ alle Knoten in C weiß. Daher gilt $f[w] > f[y]$ für alle $w \in C$.

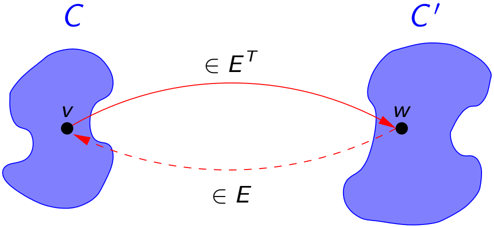
Somit: $f(C) > f(C')$.

Korrektheit (Forts.)

Korollar 2. Seien C und C' zwei verschiedene starke Zusammenhangskomponenten eines gerichteten Graphen $G = (V, E)$.

Angenommen, es gibt eine Kante $(v, w) \in E^T$, wobei $v \in C$ und $w \in C'$. Dann ist $f(C) < f(C')$.

Graph G^T:



$$\Rightarrow f(C) < f(C')$$

Korrektheit (Forts.)

Satz. $\textsc{StronglyConnectedComponents}(G)$ berechnet die starken Zusammenhangskomponenten eines gerichteten Graphen G.

Beweis durch Induktion über die Anzahl k der Depth-First-Bäume, die während der zweiten Tiefensuche gefunden werden.

Induktionsbehauptung: Die ersten k Depth-First Bäume, die während der zweiten Tiefensuche (in G^T) berechnet werden, sind starke Zusammenhangskomponenten in G.

Korrektheit (Forts.)

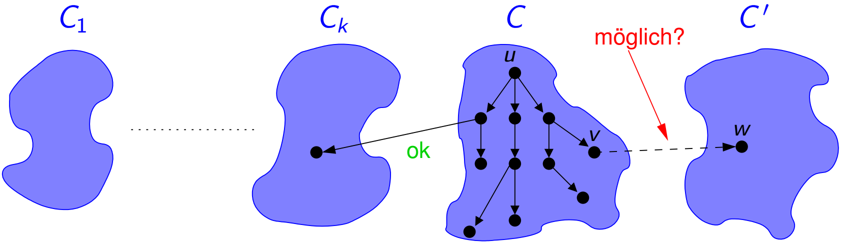
Induktionsanfang: $k = 0$ ✓

Induktionsschritt: Angenommen, in der zweiten Tiefensuche wurden bisher k Bäume gefunden und jeder dieser Bäume ist eine starke Zusammenhangskomponente.

Betrachte nun den $(k+1)$-ten Baum, der von der Tiefensuche berechnet wird. Sei u die Wurzel dieses Baums und sei C die starke Zusammenhangskomponente in G, die u enthält.

Korrektheit (Forts.)

Graph G^T:



$$f(C_1) > \ldots > f(C_k) > f(C) > f(C')$$

Beachte:

- Alle dargestellten Kanten sind in E^T
- Die f-Werte stammen von der ersten Tiefensuche

Korrektheit (Forts.)

Fakten:

- Alle Knoten in C sind auch in G^T paarweise erreichbar. Also enthält der Baum mit Wurzel u alle Knoten von C
- Wegen der Knotenauswahl in der zweiten Tiefensuche gilt für jede Zusammenhangskomponente C', die bisher noch nicht gefunden wurde, dass:

$$f[u] = f(C) > f(C')$$

Frage: Führt von einem Knoten v des Baums von C eine Kante zu einem Knoten w in einer noch nicht entdeckten Zusammenhangskomponente C'?

Korrektheit (Forts.)

Antwort: Nein!

Begründung: Zum Zeitpunkt der Entdeckung von u gilt:

- Alle Knoten in C sind weiß. Somit sind sie Nachkommen von u im entsprechenden Depth-First Baum (Weißer-Pfad-Satz)
- Wegen Korollar 2 führen alle Kanten in G^T, die C verlassen, zu einer Zusammenhangskomponente, die bereits gefunden wurde

Also sind die Knoten im Depth-First Baum von u in G^T genau die Knoten in C.

Somit wurde eine weitere starke Zusammenhangskomponente berechnet. ✓

Zusammenfassung

- Datenstrukturen für Graphen
  - Adjazenzlisten
  - Adjazenzmatrizen
- Elementare Algorithmen
  - Breitensuche
  - Tiefensuche
- Anwendungen der Tiefensuche
  - Topologische Sortierung
  - Starke Zusammenhangskomponenten
- Anwendungen der Breitensuche
  - Minimal aufspannende Bäume (Lerneinheit 6)
  - Kürzeste Wege in Graphen (Lerneinheit 7)