Informatik II - SS 2014

(Algorithmen & Datenstrukturen)

Vorlesung 15 (1.7.2014)

Graphtraversierung & Anwendungen

UNI FREIBURG

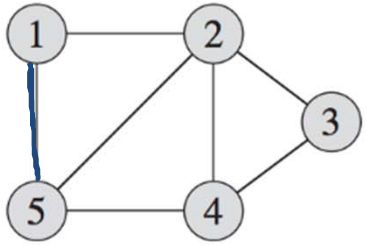
Fabian Kuhn

Algorithmen und Komplexität

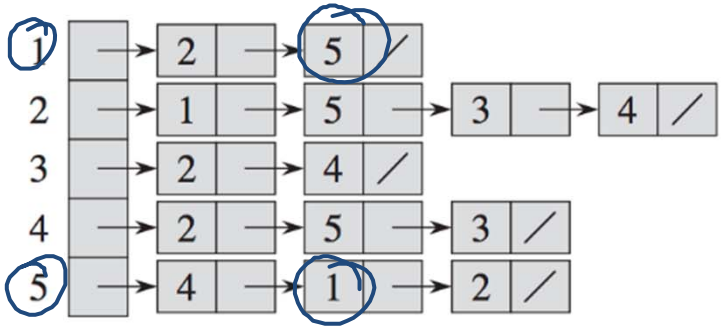
Beispiele

Beispiele aus [CLRS]:

ungerichteter Graph



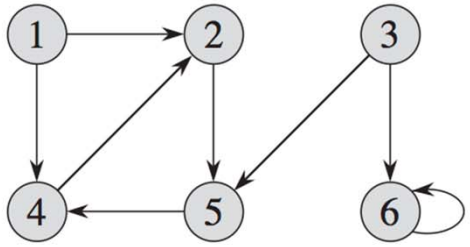
Adjazenzlisten

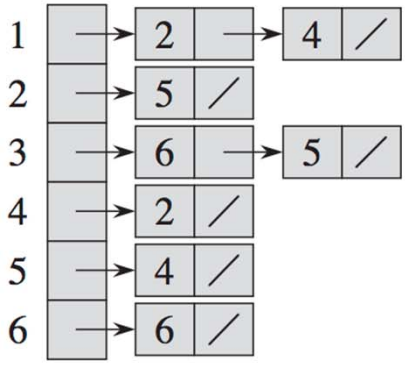


Adjazenzmatrix

	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

gerichteter





	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

Breitensuche von allgemeinen Graphen

BFS-Traversierung

Unterschiede Binärbaum T $\Leftrightarrow$ allg. Graph G

- Graph G kann Zyklen haben
- In T haben wir eine Wurzel und kennen von jedem Knoten die Richtung zur Wurzel
  - etwas allgemeiner bezeichnen wir solche Bäume auch als gewurzelte Bäume

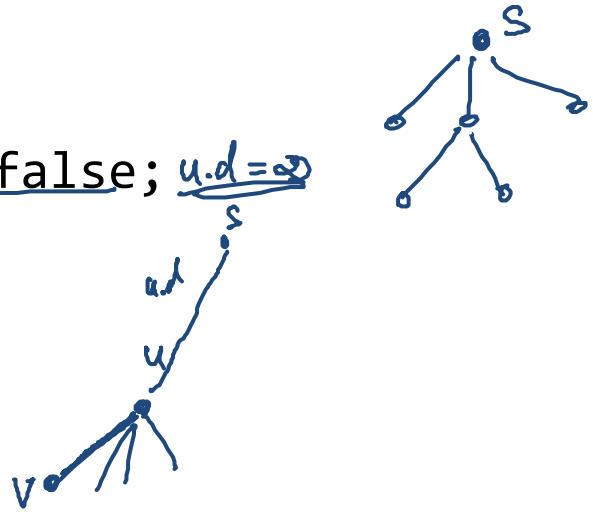
Breitensuche in Graph G (Start bei Knoten $s \in V$)

- **Zyklen: markiere** Knoten, welche man schon gesehen hat
- **Markiere** Knoten s, hänge s in die Queue
- Wie bisher, nehme immer den ersten Knoten u aus der Queue:
  - **besuche Knoten u**
  - Gehe durch die Nachbarn v von u
    Falls v nicht markiert, markiere v und hänge v in Queue
    Falls v markiert ist, muss nichts getan werden

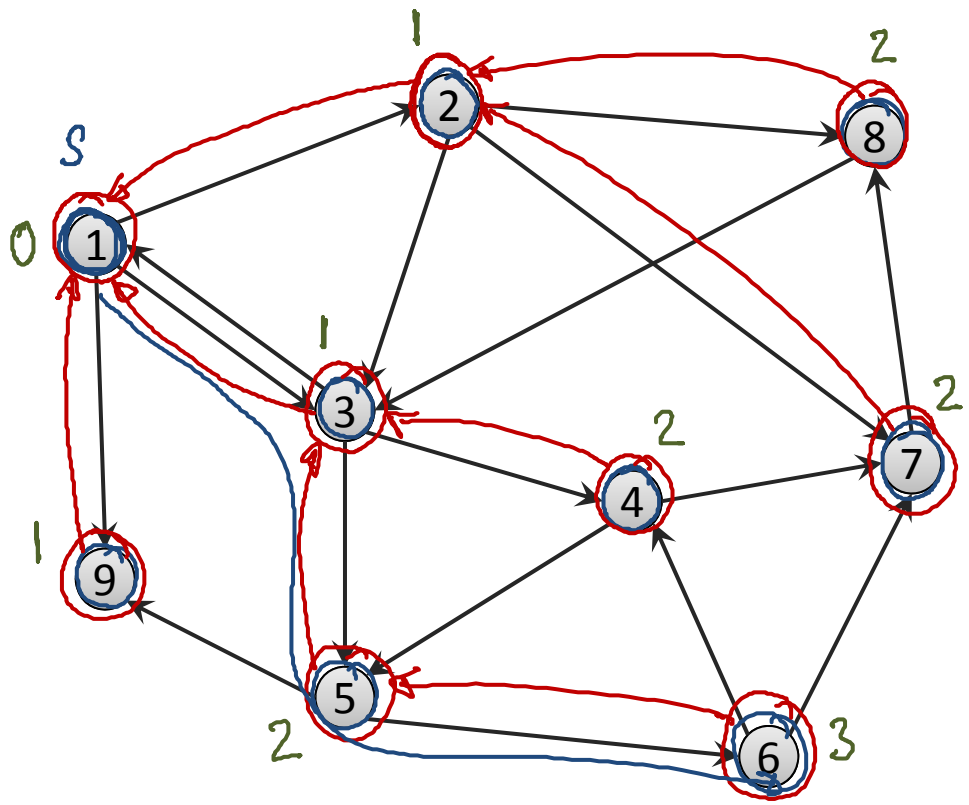
BFS Baum: Pseudocode

- Wir merken uns zusätzlich die Distanz zu s im Baum

```
BFS-Tree:
    Q = new Queue();
    for all u in V: u.marked = false;
    root.marked = true;
    root.parent = NULL;
    root.d = 0
    Q.enqueue(root)
    while not Q.empty() do
        u = Q.dequeue()
        visit(u)
        for v in u.neighbors do
            if not v.marked then
                v.marked = true;
                v.parent = u;
                v.d = u.d + 1;
                Q.enqueue(v)
```



Breitensuche Beispiel

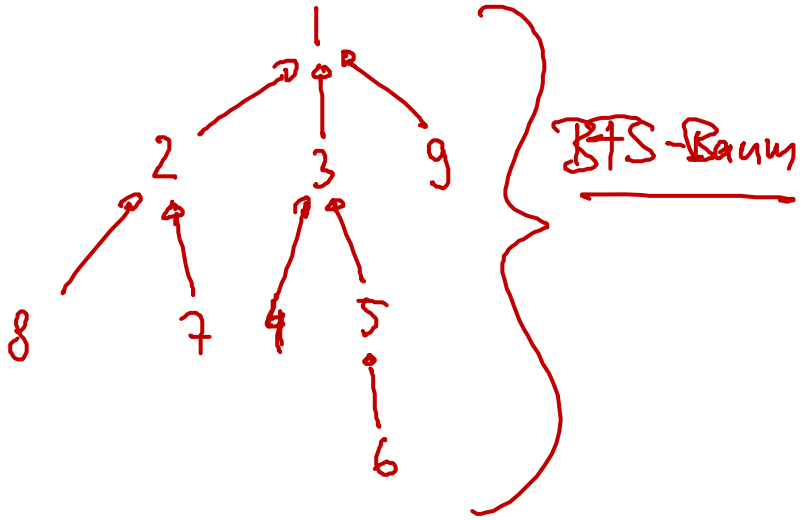


Queue Q:

~~8~~, ~~2~~, ~~3~~, ~~9~~, ~~7~~, ~~8~~, ~~4~~, ~~5~~, ~~6~~

BFS-Reihenfolge:

1, 2, 3, 9, 7, 8, 4, 5, 6



Laufzeit: $O(m+n)$

m: #Kanten, n: #Knoten

Analyse Breitensuche

UNI FREIBURG

In der Folge benennen wir die Knoten folgendermaßen

- weiße Knoten: Knoten, welche der Alg. noch nicht gesehen hat
- graue Knoten: markierte Knoten
  - Knoten werden grau, wenn sie in die Warteschlange eingefügt werden
  - Knoten sind grau, solange sie in der Warteschlange sind
- schwarze Knoten: besuchte Knoten
  - Knoten werden schwarz, wenn sie aus der Warteschlange genommen werden

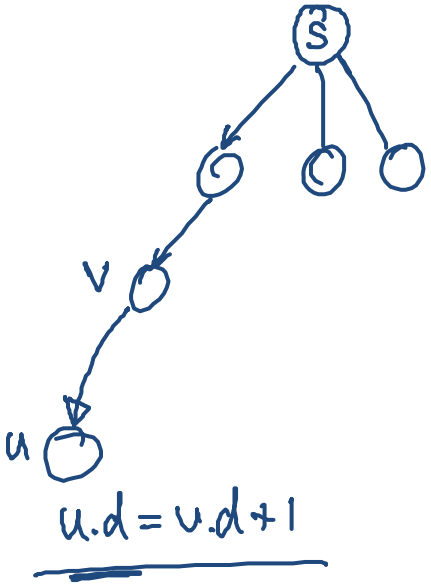
Die Laufzeit der BFS-Traversierung ist $O(n+m)$.

Analyse Breitensuche

Im BFS-Baum eines ungewichteten Graphen ist die Distanz von jedem Knoten u zur Wurzel s gleich $d_G(s,u)$.

- Baumdistanz zur Wurzel: $d_T(s,u) = u.d$
- Wir müssen also zeigen, dass $u.d = d_G(s,u)$
- Wir zeigen zuerst, dass $\boldsymbol{u.d \geq d_G(s,u)}$

$u.d = \infty$



Analyse Breitensuche

Lemma: Annahme: Während BFS-Traversal ist Zustand der Queue

$$Q = \langle v_1, v_2, \dots, v_r \rangle \quad (v_1\text{: head}, v_r\text{: tail})$$

Dann gilt $v_r.d \leq v_1.d + 1$ und $v_i.d \leq v_{i+1}.d$ (für $i = 1, \dots, r-1$)

Beweis:

Per Induktion über die Warteschlangenoperationen

dequeue-Operation:

$v_1, v_2, \dots, v_r$

$v_1.d \leq v_2.d \leq v_r.d$

$v_r.d \leq v_1.d + 1 \leq v_2.d + 1$

enqueue-Operation:

$(u)\ v_1, v_2, \dots, v_r, v$

$u.d \leq v_1.d$

$v.d = u.d + 1 \leq v_1.d + 1$

Algor.

$v_r.d \leq u.d + 1$

$v.d$

Insbesondere: Falls v_i vor v_j in Q eingefügt wird, dann $v_i.d \leq v_j.d$

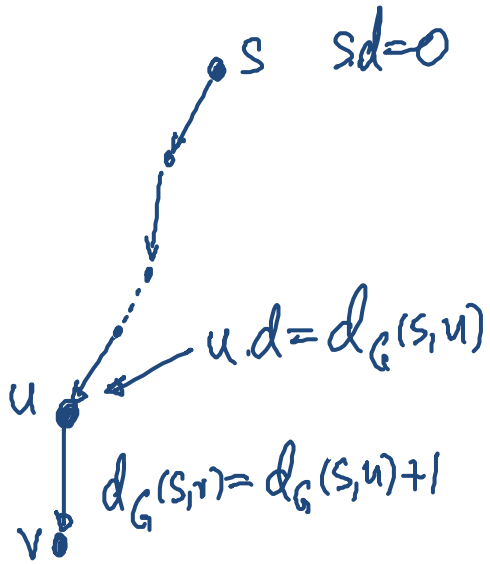
Analyse Breitensuche

shortest path tree

Im BFS-Baum eines ungewichteten Graphen ist die Distanz von jedem Knoten u zur Wurzel s gleich $d_G(s,u)$.

Widerspruchsbeweis

Annahme $\exists v \in V: v.d > d_G(s,v)$, v ist ein solcher Knoten mit minimalem $d_G(s,v) \geq 1$



$v.d > d_G(s,v) = d_G(s,u)+1 = u.d+1$

dequeue von u

wir betrachten v

v ist weiss $\Rightarrow v.d = u.d+1$ (kann nicht sein)

v ist markiert:

v schwarz $v.d \leq u.d$ (" in ")

v grau : v ist in Q

Lemma: $v.d \leq u.d+1$ (" " ")

Analyse Breitensuche

UNI FREIBURG

Im BFS-Baum eines ungewichteten Graphen ist die Distanz von jedem Knoten u zur Wurzel s gleich $d_G(s,u)$.

Tiefensuche in allgemeinen Graphen

UNI FREIBURG

Grundidee Tiefensuche in G (Start bei Knoten $s \in V$)

- **Markiere Knoten v** (am Anfang ist $v = s$)
- Besuche die Nachbarn von v der Reihe nach *rekursiv*
- Nachdem alle Nachbarn besucht sind, **besuche s**

- **rekursiv:** Beim Besuchen der Nachbarn werden deren Nachbarn besucht, und dabei deren Nachbarn, etc.
- **Zyklen in G:** Besuche jeweils nur Knoten, welche noch nicht markiert sind

- entspricht der Postorder-Traversierung in Bäumen
- Fall man gleich beim Markieren den Knoten besucht, entspricht es der Preorder-Traversierung

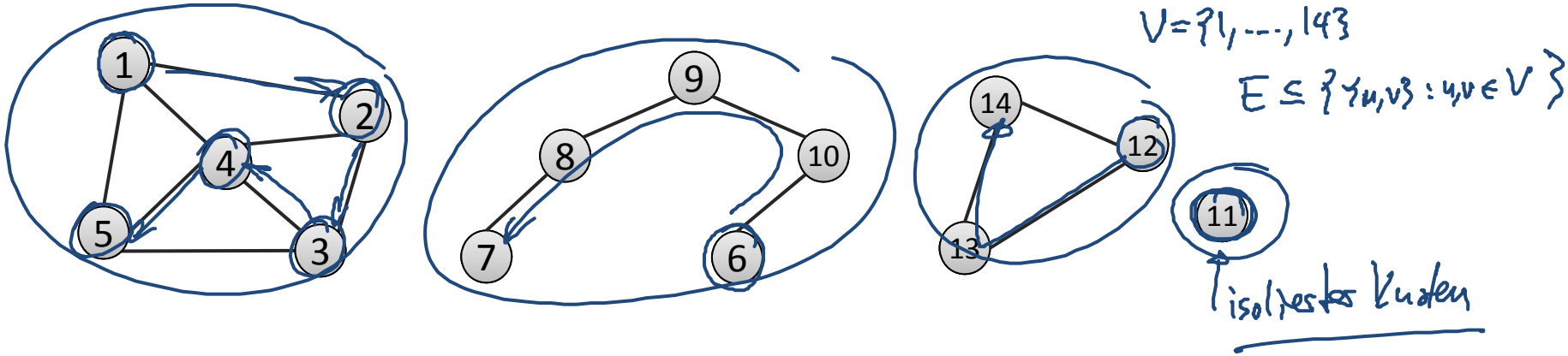
Tiefensuche: Pseusocode

```
DFS-Traversal:
    for all u in V: u.color = white;
    DFS-visit(root, NULL)

DFS-visit(u, p):
    u.color = gray;
    u.parent = p;
    for all v in u.neighbors do
        if v.color = white
            DFS-visit(v, u)
    visit node u;
    u.color = black;
```

Zusammenhangskomponenten

- Die Zusammenhangskomponenten (oder einfach Komponenten) eines Graphen sind seine zusammenhängenden Teile.



Ziel: Finde alle Komponenten eines Graphen.

```
for u in V do
    if not u.marked then
        start new component
        explore with DFS/BFS starting at u
```

- Die Zusammenhangskomponenten eines Graphen können in $O(n+m)$ Zeit identifiziert werden. (mit Hilfe von DFS oder BFS)

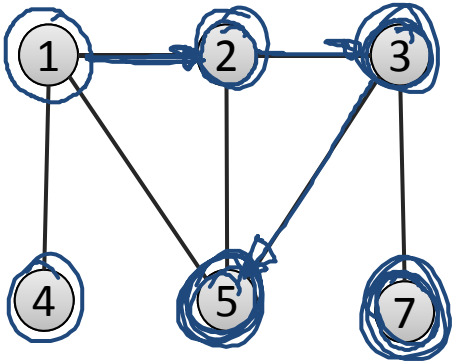
DFS- “Klammer”-Theorem

Wir definieren für jeden Knoten v die folgenden zwei Zeitpunkte

- $t_{v,s}$: Zeitpunkt, wenn v in der DFS-Suche grau gefärbt wird
- $t_{v,f}$: Zeitpunkt, wenn v in der DFS-Suche schwarz gefärbt wird

Theorem: Im DFS-Baum ist ein Knoten v ist genau dann im Teilbaum eines Knoten u, falls das Intervall $[t_{v,s}, t_{v,f}]$ vollständig im Intervall $[t_{u,s}, t_{u,f}]$ enthalten ist.

Beispiel:



$t_{1,s}, t_{2,s}, t_{3,s}, t_{5,s}, t_{5,f}, t_{7,s}, t_{7,f}, t_{3,f}, t_{2,f}, t_{4,s}, t_{4,f}, t_{1,f}$

DFS- “Klammer”-Theorem

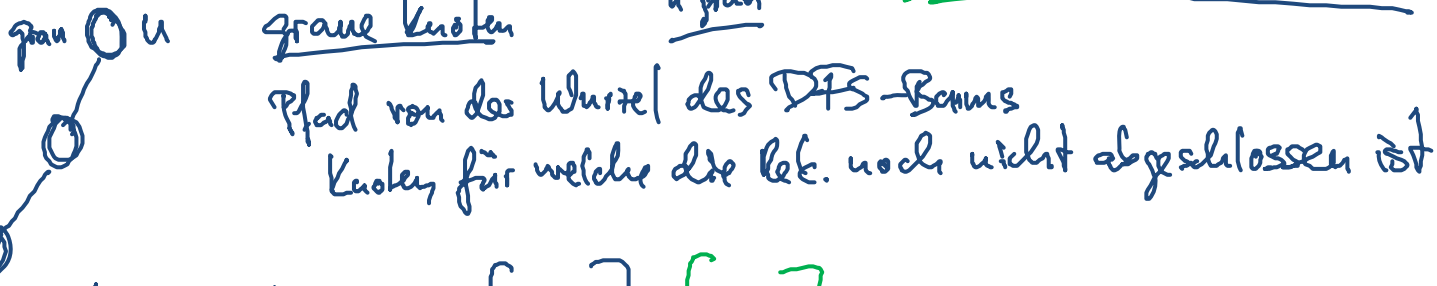
Theorem: Im DFS-Baum ist ein Knoten v ist genau dann im Teilbaum eines Knoten u, falls das Intervall $[t_{v,s}, t_{v,f}]$ vollständig im Intervall $[t_{u,s}, t_{u,f}]$ enthalten ist.

- Insbesondere sind zwei Intervalle entweder disjunkt, oder das eine ist komplett im anderen enthalten.

Beweis:

$t_{u,s} < t_{v,s}$

Fall 1: $t_{v,s} < t_{u,f}$ ✓



graue Knoten

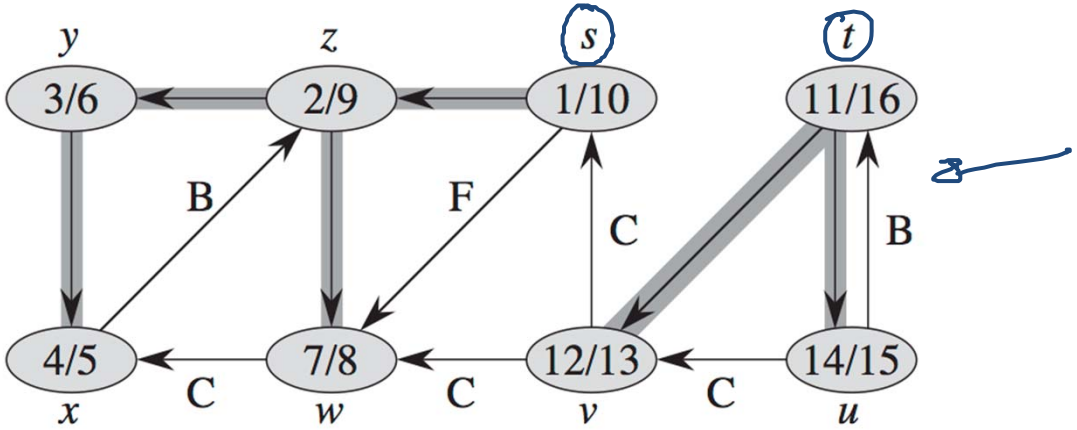
Pfad von der Wurzel des DFS-Baums

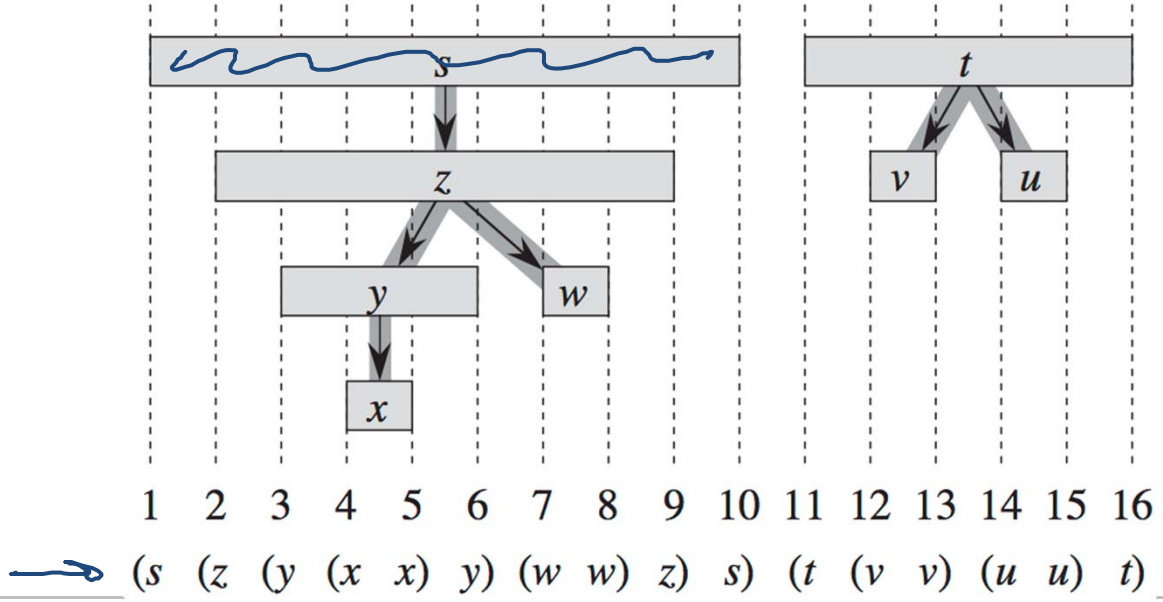
Knoten für welche die Rek. noch nicht abgeschlossen ist

Fall 2: $t_{v,s} > t_{u,f}$

DFS- “Klammer”-Theorem

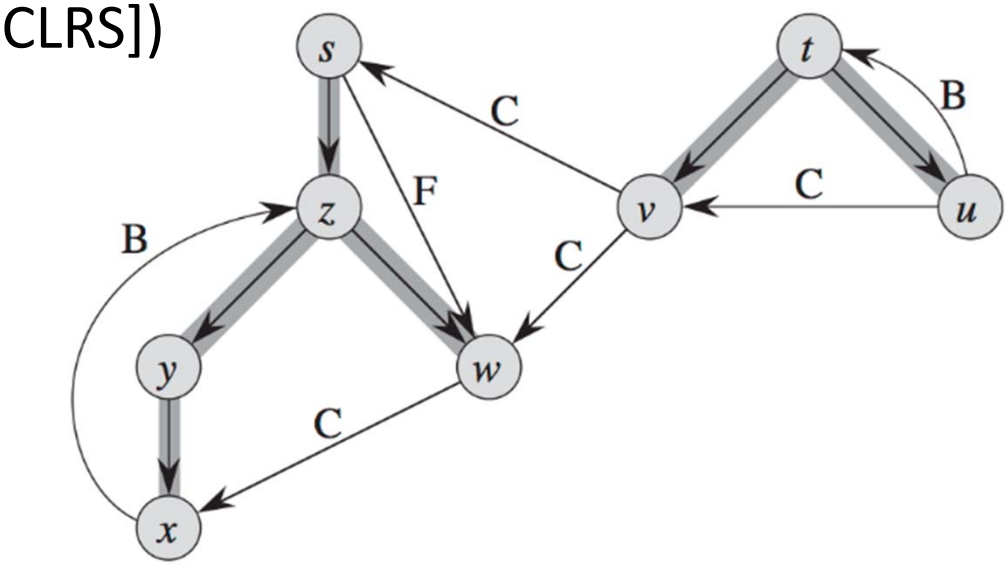
Beispiel (aus [CLRS])

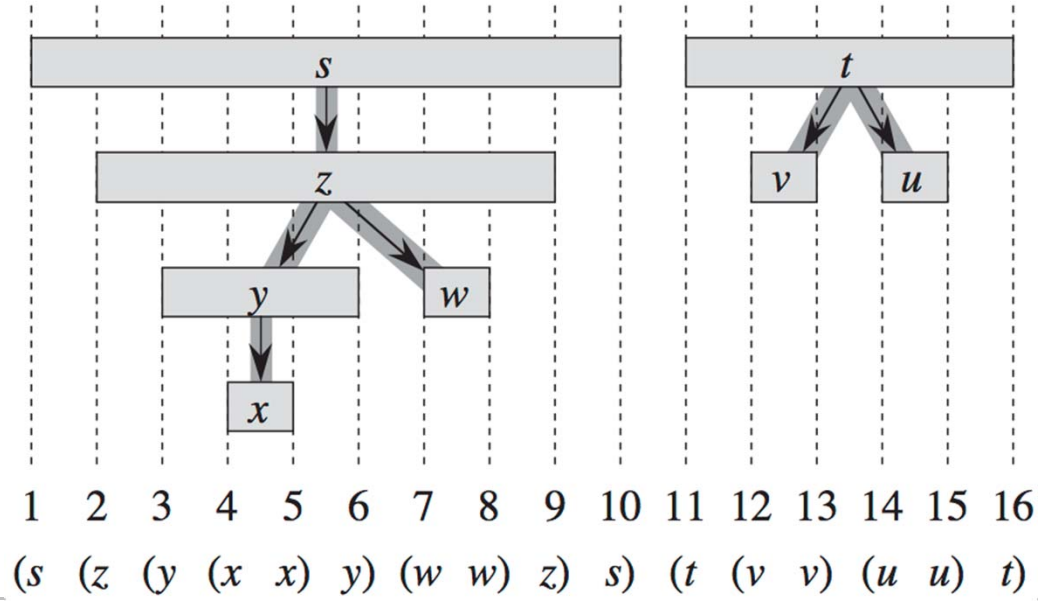




DFS- “Klammer”-Theorem

Beispiel (aus [CLRS])





DFS- “Klammer”-Theorem

UNI FREIBURG

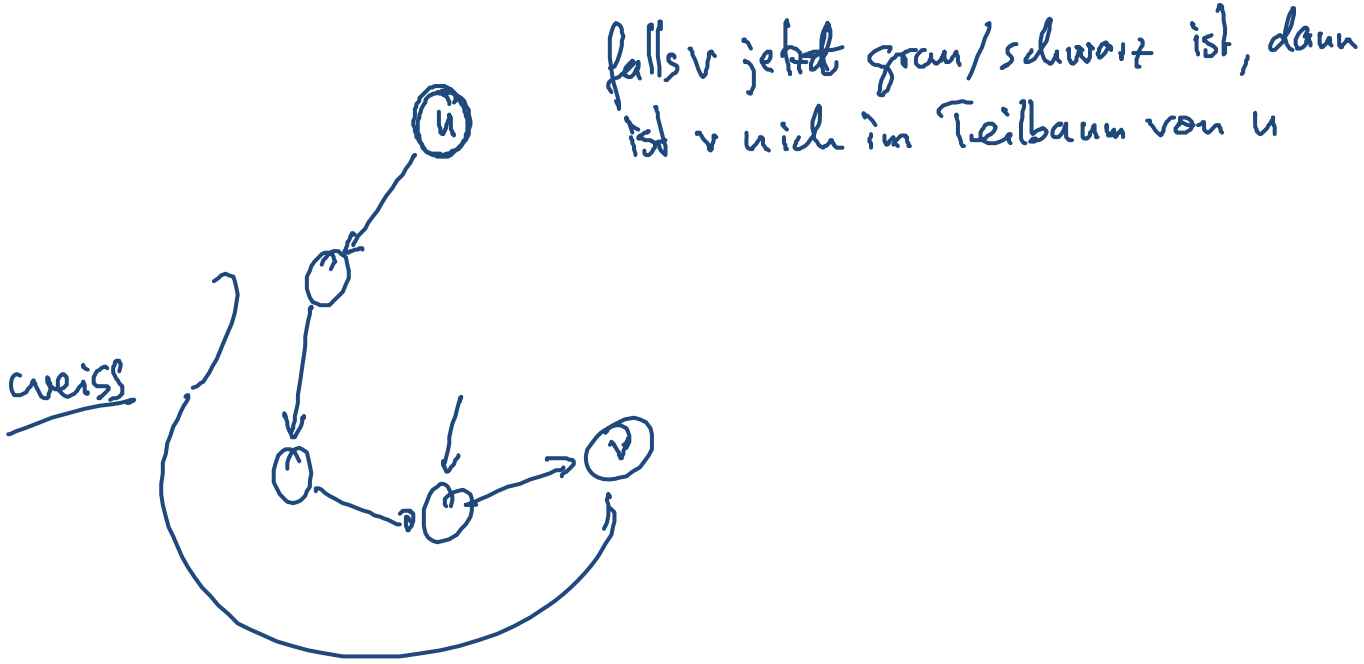
Theorem: Im DFS-Baum ist ein Knoten v ist genau dann im Teilbaum eines Knoten u, falls das Intervall $[t_{v,s}, t_{v,f}]$ vollständig im Intervall $[t_{u,s}, t_{u,f}]$ enthalten ist.

Implikationen

- Zwei Intervalle sind entweder disjunkt, oder das eine ist komplett im anderen enthalten.
- Ein weisser Knoten v, welcher in der rekursiven Suche von u entdeckt wird, wird schwarz, bevor die Rekursion zu u zurückkehrt.
- Wieso “Klammer”-Theorem:
  Wenn man bei jedem $t_{v,1}$ eine öffnende Klammer und bei jedem $t_{v,2}$ eine schließende Klammer hinschreibt, bekommt man ein Klammerausdruck, welcher korrekt geschachtelt ist.

Weiße Pfade

Theorem: In einem DFS-Baum ist ein Knoten v genau dann im Teilbaum eines Knoten u, falls unmittelbar vor dem Markieren von u, ein komplett weißer Pfad von u nach v besteht.



Klassifizierung der Kanten (bei DFS-Suche)

Baumkanten:

- (u, v) ist eine Baumkante, falls v von u aus entdeckt wird

Rückwärtskanten:

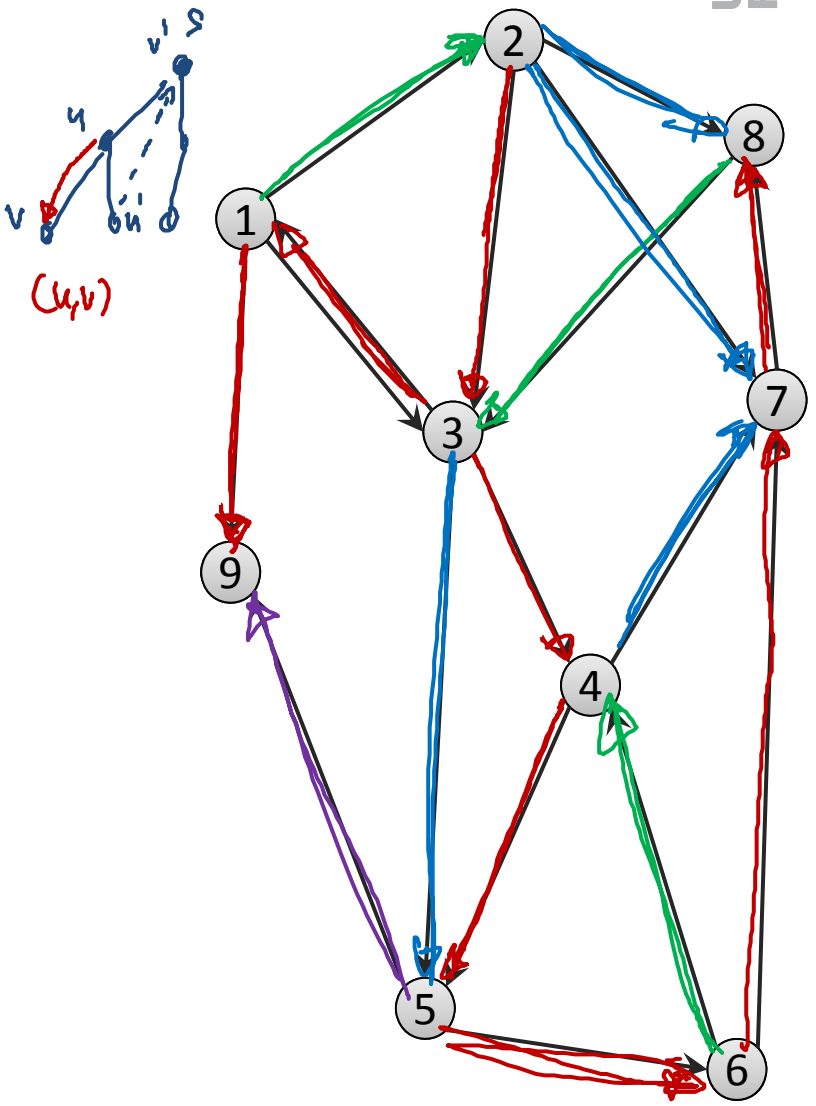
- (u, v) ist eine Rückwärtskante, falls v eine Vorgängerknoten von u ist

Vorwärtskanten:

- (u, v) ist eine Vorwärtskante, falls v ein Nachfolgerknoten von u ist

Querkanten:

- Alle übrigen Kanten



Klassifizierung der Kanten (bei DFS-Suche)

Baumkante (u, v):

Bei Betrachtung von (u,v)

falls v weiss ist $\Longleftrightarrow$ (u,v) Baumkante

Rückwärtskante (u, v):

Bei Betr. von (u,v)

v ist grau $\Longleftrightarrow$ Rückwärtskante

Vorwärtskante (u, v):

Bei Betr. von (u,v)

v schwarz

Vorwärtsk.: $t_{u,s} < t_{v,s}$

Querkante (u, v):

Querkante: $t_{u,s} > t_{v,s}$

DFS – Ungerichtete Graphen

UNI FREIBURG

Theorem: Bei einer DFS-Suche in ungerichteten Graphen ist jede Kante entweder eine Baumkante oder eine Rückwärtskante.