

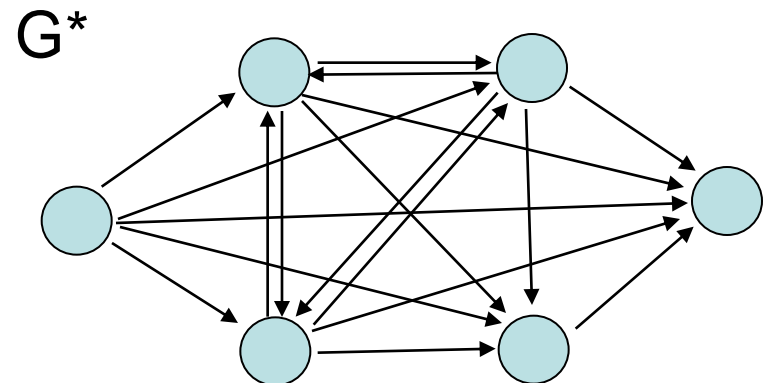
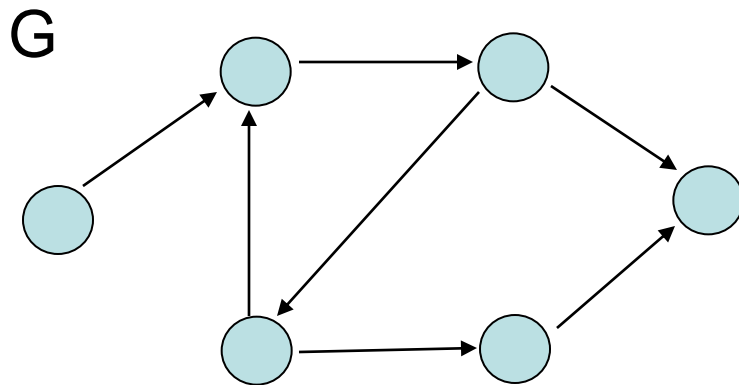


## Datenstrukturen, Algorithmen und Programmierung 2 (DAP2)

## Graphalgorithmen

### *Das transitive Hülle Problem*

- Gegeben sei ein gerichteter, ungewichteter Graph  $G=(V,E)$
- Gesucht: Die transitive Hülle  $G^*=(V,E^*)$  von  $G$ , wobei  $E^*=\{(u,v): \text{es gibt Weg von } u \text{ nach } v \text{ in } G\}$



## Graphalgorithmen

### *Transitive Hülle*

- in  $O(|V|^3)$  Zeit mit Floyd-Warshall
- In  $O(|V|^2 + |V| \cdot |E|)$  Zeit mit Breiten- oder Tiefensuche von jedem Knoten
- Geht das auch schneller?

## Graphalgorithmen

### *Graphen und Matrixmultiplikation*

- Sei  $A$  die  $n \times n$ -Adjazenzmatrix eines ungerichteten Graphen  $G$  mit Knotenmenge  $\{1, \dots, n\}$
- Was ist  $A \cdot A$ ?

## Graphalgorithmen

### *Behauptung 62*

Sei  $Z = A \cdot A$ . Dann gilt  $z_{ij} > 0$ , g.d.w. es in  $G$  einen Pfad der Länge 2 von Knoten  $i$  zu Knoten  $j$  gibt.

## Graphalgorithmen

### *Behauptung 63*

Sei  $Z' = A \cdot A + A$ . Dann gilt, dass  $z'_{ij} > 0$ , g.d.w. es einen Weg der Länge 1 oder 2 von Knoten  $i$  zu Knoten  $j$  gibt.

### *Beweis*

- Folgt sofort aus dem vorhergehenden Lemma und der Tatsache, dass  $A$  genau für die Paare  $i, j$  einen Eintrag 1 hat, die durch eine Kante (einen Weg der Länge 1) verbunden sind

### *Konstruiere Matrix $B$ mit:*

- $b_{ij} = 1 \Leftrightarrow z'_{ij} > 0$

## Graphalgorithmen

### *Behauptung 64*

Der durch Matrix B definierte Graph hat einen Weg von Knoten i nach j, g.d.w.  
der durch Matrix A definierte Graph einen solchen Weg hat.

### *Beweis*

Folgt aus der vorherigen Behauptung.

## Graphalgorithmen

### *Behauptung 65*

Sei  $P$  ein Weg der Länge  $k > 1$  in  $G$  von Knoten  $i$  zu Knoten  $j$ . Dann gibt es in dem von Matrix  $B$  beschriebenen Graphen  $G'$  einen Weg von  $i$  nach  $j$  mit Länge maximal  $\frac{2}{3}k$ .

### *Beweis*

- Sei  $P$  ein Weg in  $G$  mit  $k$  Kanten.
- Ist  $k$  gerade, dann gibt es in  $G'$  einen Weg der Länge  $k/2$ , da man jeweils zwei aufeinanderfolgende Kanten von  $P$  durch eine Kante in  $G'$  ersetzen kann.
- Ist  $k$  ungerade, dann gibt es in  $G'$  einen Weg der Länge  $\lceil k/2 \rceil$ , da man bis auf die letzte Kante jeweils zwei aufeinanderfolgende Kanten von  $P$  durch eine Kante in  $G'$  ersetzen kann.
- Somit gilt für  $k=3$ , dass die Länge des Weges in  $G'$   $\frac{2}{3}$  der Länge des Weges in  $G$  ist. Für  $k > 3$  verkürzt sich die Weglänge entsprechend mehr.

## Graphalgorithmen

### *Konsequenz aus Beh. 64 und 65*

- Wenn wir die Berechnung von  $B \log_{3/2} n$  mal iterieren, haben wir die transitive Hülle berechnet

## Graphalgorithmen

TransitiveHülle(A)

1. **for**  $i \leftarrow 1$  **to**  $\log_{3/2} n$  **do**
2.      $Z' \leftarrow A A+A$
3.     **for**  $i \leftarrow 1$  **to**  $n$  **do**
4.         **for**  $j \leftarrow 1$  **to**  $n$  **do**
5.             **if**  $z_{ij} > 0$  **then**  $b_{ij} \leftarrow 1$  **else**  $b_{ij} \leftarrow 0$
6.      $A \leftarrow B$
7. **return** A

Laufzeit

- $O(\log n)$   
 $O(M(n) \log n)$   
 $O(n \log n)$   
 $O(n^2 \log n)$   
 $O(n^2 \log n)$   
 $O(n^2 \log n)$   
 $O(1)$

$M(n)$ : Zeit um zwei  $n \times n$ -Matrizen zu multiplizieren

## Graphalgorithmen

### Satz 66

Der Algorithmus TransitiveHülle berechnet die transitive Hülle eines Graphen  $G$  in  $O(M(n) \log n)$  Zeit, wobei  $M(n)$  die Laufzeit zur Multiplikation zweier  $n \times n$ -Matrizen bezeichnet.

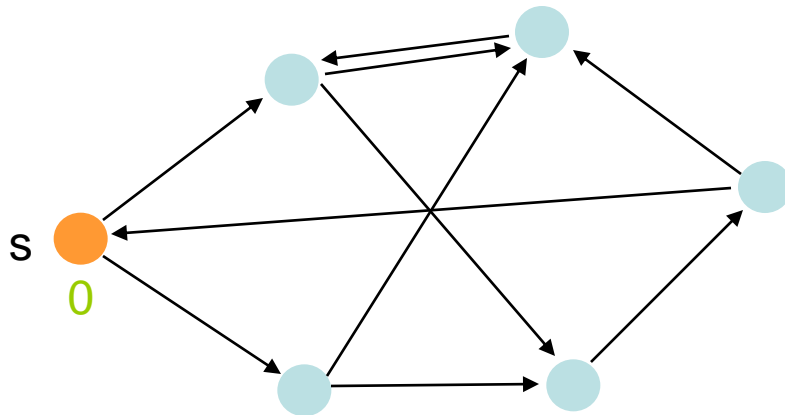
## Graphalgorithmen

### *Breitensuche*

- Durchlauf verbundenen Graph von Startknoten  $s$
- Berechne kürzeste Wege (Anzahl Kanten) von  $s$  zu anderen Knoten im Graph
- Eingabegraph in Adjazenzlistendarstellung
- Laufzeit  $O(|V|+|E|)$

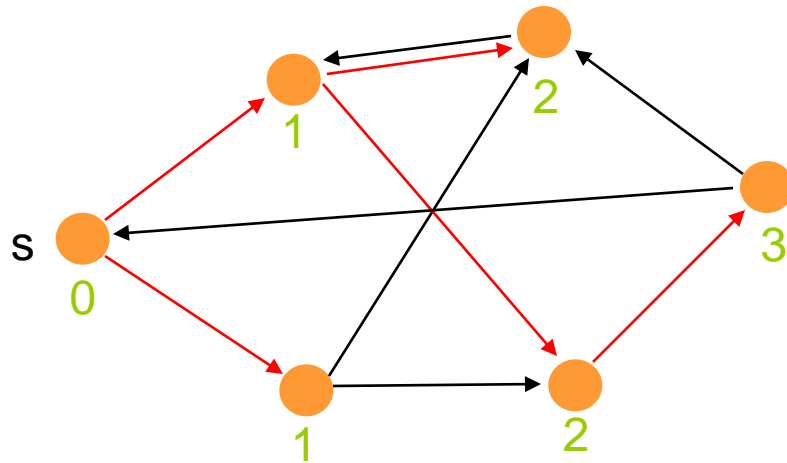
## Graphalgorithmen (siehe Vollversion)

### Breitensuche



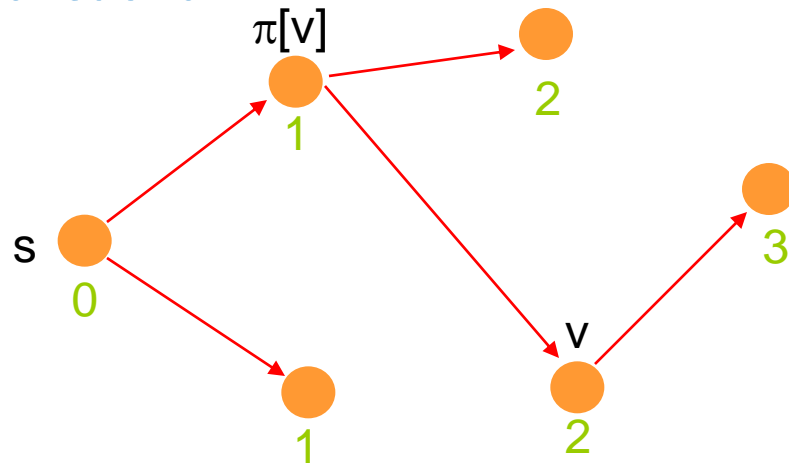
## Graphalgorithmen (siehe Vollversion)

### Breitensuche



## Graphalgorithmen

### Breitensuche



Vorgängergraph  $G$  enthält alle Kanten  $(\pi[v], v)$ .

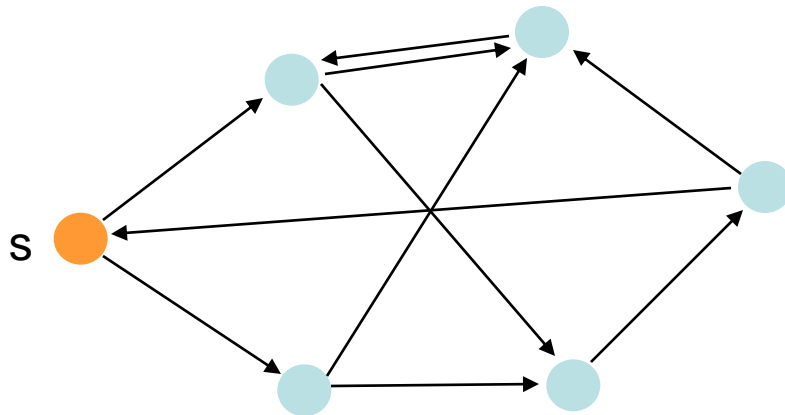
## Graphalgorithmen

### *Tiefensuche*

- Suche zunächst „tiefer“ im Graph
- Neue Knoten werden immer vom zuletzt gefundenen Knoten entdeckt
- Sind alle adjazenten Knoten des zuletzt gefundenen Knoten  $v$  bereits entdeckt, springe zurück zum Knoten, von dem aus  $v$  entdeckt wurde
- Wenn irgendwelche unentdeckten Knoten übrigbleiben, starte Tiefensuche von einem dieser Knoten

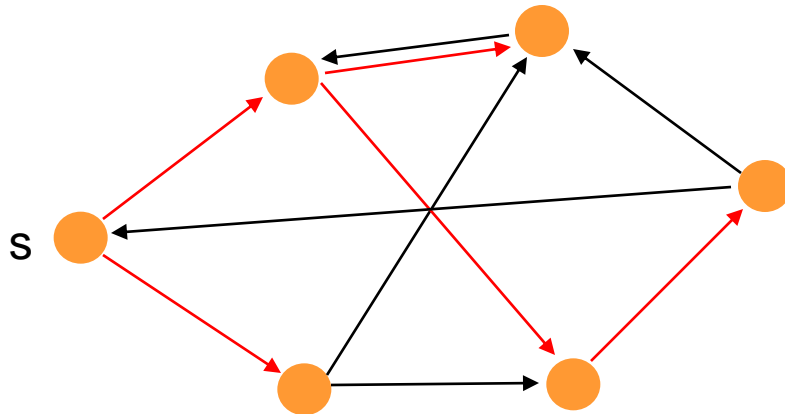
## Graphalgorithmen (siehe Vollversion)

### Tiefensuche



## Graphalgorithmen (siehe Vollversion)

### Tiefensuche



## Graphalgorithmen

### *Invariante Tiefensuche*

- Zu Beginn: alle Knoten weiß
- Entdeckte Knoten werden grau
- Abgearbeitete Knoten werden schwarz
- Zwei Zeitstempel:  $d[v]$  und  $f[v]$  (liegen zwischen 1 und  $2|V|$ )
- $d[v]$ :  $v$  ist entdeckt
- $f[v]$ :  $v$  ist abgearbeitet

## Graphalgorithmen

### *Zeitstempel der Tiefensuche*

- $d[v] < f[v]$
- Vor  $d[v]$  ist  $v$  weiß
- Zwischen  $d[v]$  und  $f[v]$  ist  $v$  grau
- Nach  $f[v]$  ist  $v$  schwarz

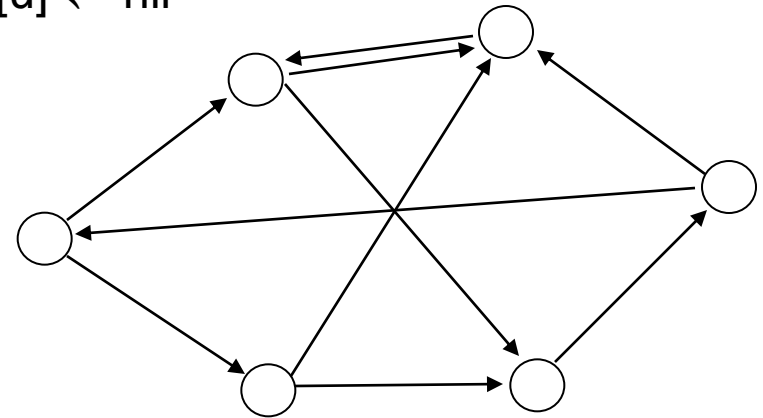
## Graphalgorithmen (siehe Vollversion)

### DFS(G)

1. **for each** vertex  $u \in V$  **do**  $\text{color}[u] \leftarrow \text{weiß}$  ;  $\pi[u] \leftarrow \text{nil}$
2.  $\text{time} \leftarrow 0$
3. **for each** vertex  $u \in V$  **do**
4.   **if**  $\text{color}[u] = \text{weiß}$  **then** DFS-Visit( $u$ )

### DFS-Visit( $u$ )

1.  $\text{color}[u] \leftarrow \text{grau}$
2.  $\text{time} \leftarrow \text{time} + 1$ ;  $d[u] \leftarrow \text{time}$
3. **for each**  $v \in \text{Adj}[u]$  **do**
4.   **if**  $\text{color}[v] = \text{weiß}$  **then**  $\pi[v] \leftarrow u$  ; DFS-Visit( $v$ )
5.  $\text{color}[u] \leftarrow \text{schwarz}$
6.  $\text{time} \leftarrow \text{time} + 1$ ;  $f[u] \leftarrow \text{time}$





## Graphalgorithmen

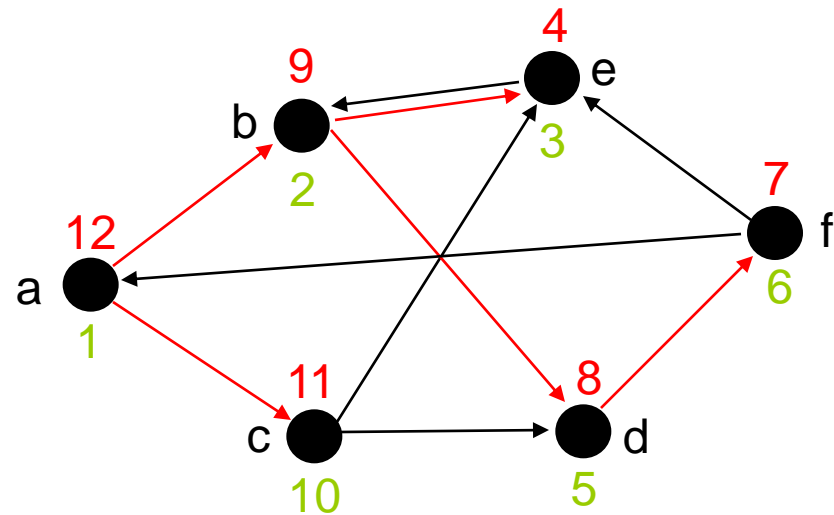
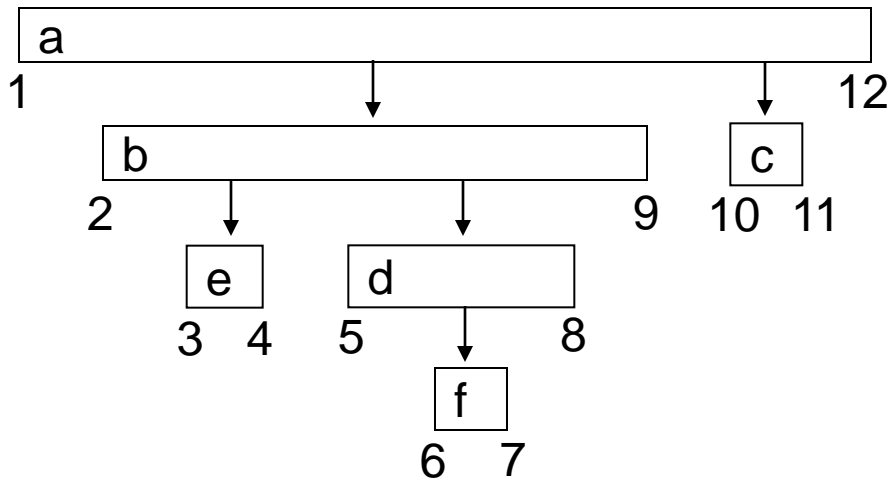
### *Satz 67 (Klammersatz zur Tiefensuche):*

In jeder Tiefensuche eines gerichteten oder ungerichteten Graphen gilt für jeden Knoten  $u$  und  $v$  genau eine der folgenden drei Bedingungen:

- Die Intervalle  $[d[u], f[u]]$  und  $[d[v], f[v]]$  sind vollständig disjunkt
- Intervall  $[d[u], f[u]]$  ist vollständig im Intervall  $[d[v], f[v]]$  enthalten und  $u$  ist Nachfolger von  $v$  im DFS-Baum
- Intervall  $[d[v], f[v]]$  ist vollständig im Intervall  $[d[u], f[u]]$  enthalten und  $v$  ist Nachfolger von  $u$  im DFS-Baum

# Graphalgorithmen

## Beispiel



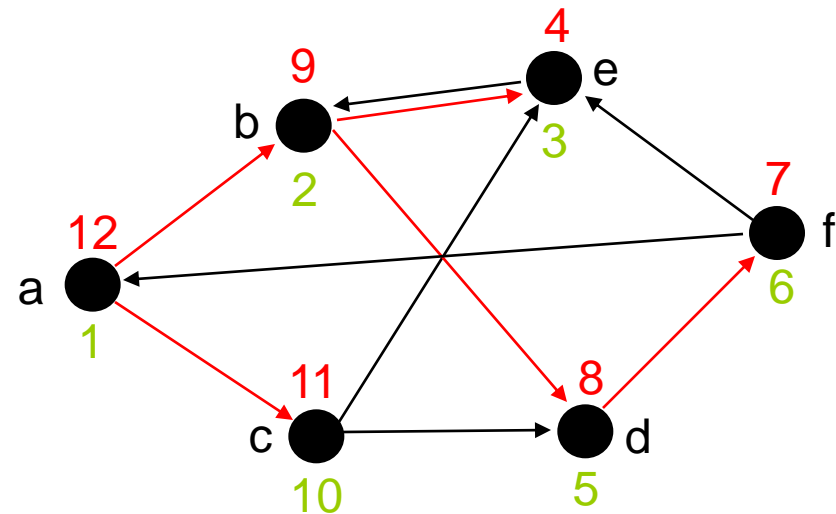
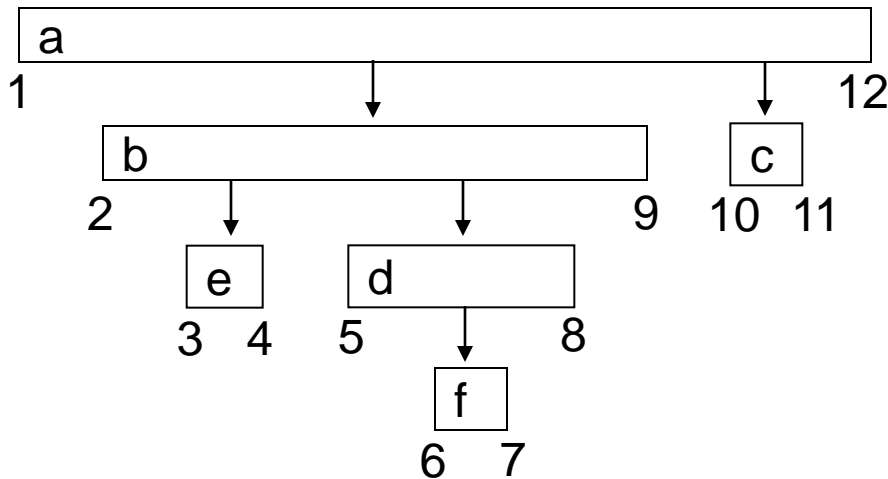
## Graphalgorithmen

### *Beweis*

- Fall 1:  $d[u] < d[v]$ .
- (a)  $d[v] < f[u]$ :
  - $v$  wurde entdeckt als  $u$  noch grau war.
  - $\Rightarrow v$  Nachfolger von  $u$
  - Da  $v$  nach  $u$  entdeckt wurde, werden alle seine ausgehenden Kanten entdeckt und wird  $v$  abgearbeitet bevor die Suche zu  $u$  zurückkehrt und  $u$  abarbeitet
  - Daher ist  $[d[v], f[v]]$  in  $[d[u], f[u]]$  enthalten
- (b)  $f[u] < d[v]$ : Dann sind die Intervalle disjunkt
- Fall 2: analog

# Graphalgorithmen

## Beispiel



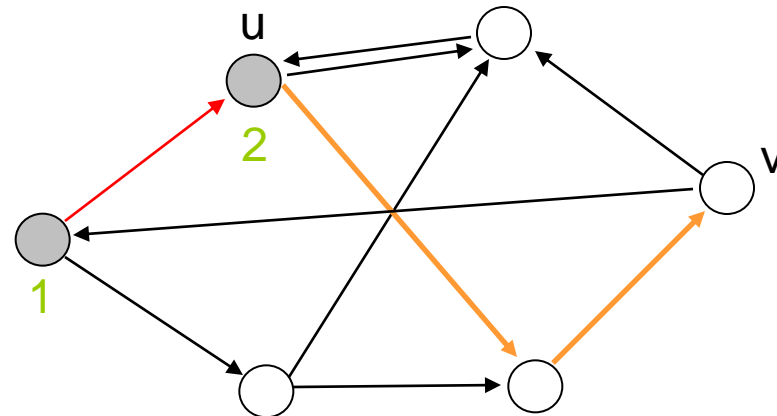
## Korollar 68

Knoten  $v$  ist echter Nachfolger von Knoten  $u$  im DFS-Baum von  $G$ , gdw.  
 $d[u] < d[v] < f[v] < f[u]$ .

## Graphalgorithmen

### Satz 69 (Satz vom weißen Weg)

In einem DFS-Wald eines gerichteten oder ungerichteten Graph  $G$  ist Knoten  $v$  ein Nachfolger von Knoten  $u$ , gdw. Zum Zeitpunkt  $d[u]$   $v$  über einen Pfad weißer Knoten erreicht werden kann.



## Graphalgorithmen

### *Satz 69 (Satz vom weißen Weg)*

In einem DFS-Wald eines gerichteten oder ungerichteten Graph  $G$  ist Knoten  $v$  ein Nachfolger von Knoten  $u$ , gdw. Zum Zeitpunkt  $d[u]$   $v$  über einen Pfad weißer Knoten erreicht werden kann.

### *Beweis*

„ $\Rightarrow$ “ Annahme:  $v$  Nachfolger von  $u$  im DFS Wald

- Sei  $w$  beliebiger Knoten auf Pfad im DFS Baum von  $u$  nach  $v$
- Damit ist  $w$  Nachfolger von  $u$
- Nach Korollar 68:  $d[u] < d[w]$ . Somit ist  $w$  weiss

## Graphalgorithmen

### *Satz 69 (Satz vom weißen Weg)*

In einem DFS-Wald eines gerichteten oder ungerichteten Graph  $G$  ist Knoten  $v$  ein Nachfolger von Knoten  $u$ , gdw. Zum Zeitpunkt  $d[u]$   $v$  über einen Pfad weißer Knoten erreicht werden kann.

### *Beweis*

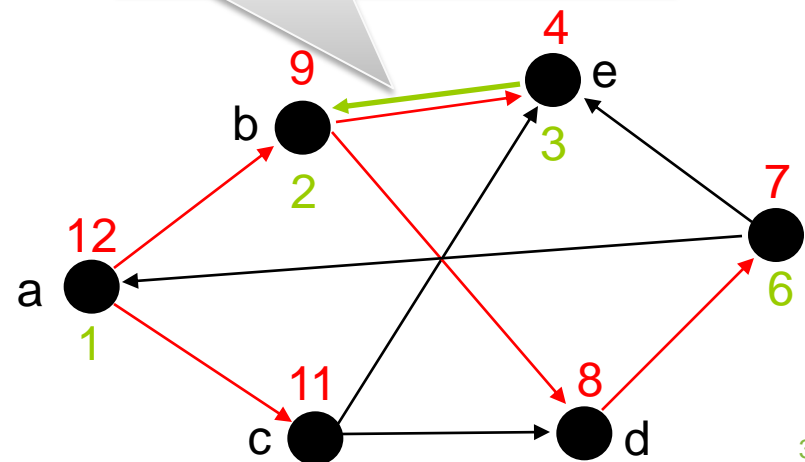
„ $\Leftarrow$ “ Annahme:  $v$  ist erreichbar von  $u$  über Pfad aus weißen Knoten zum Zeitpunkt  $d[u]$ , aber  $v$  wird nicht Nachfolger von  $u$  im DFS Wald

- ObdA. sei  $v$  der einzige Knoten auf Pfad, der nicht Nachfolger wird
- Sei  $w$  der Vorgänger von  $v$  auf dem Pfad und sei  $w$  Nachfolger von  $u$
- Korollar 68:  $f[w] \leq f[u]$
- $v$  muss entdeckt werden nachdem  $u$  entdeckt wurde und bevor  $w$  abgearbeitete ist, d.h.  $d[u] < d[v] < f[w] \leq f[u]$
- Somit ist  $[d[v], f[v]]$  in  $[d[u], f[u]]$  enthalten (Satz 67) und nach Korollar 68 ist  $v$  Nachfolger von  $u$

# Graphalgorithmen

## Klassifikation von Kanten

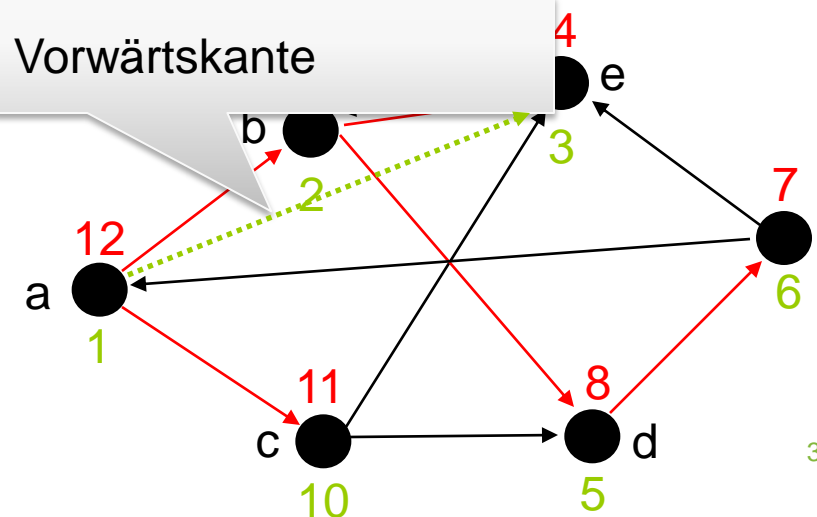
- **Baumkanten** sind Kanten des DFS-Walds  $G$
- **Rückwärtskanten** sind Kanten  $(u,v)$ , die Knoten  $u$  mit Vorgängern von  $u$  im DFS-Baum verbinden
- **Vorwärtskanten** sind die nicht-Baum Kanten  $(u,v)$ , die  $u$  mit einem Nachfolger  $v$  in einem DFS-Baum verbinden
- **Kreuzungskanten** sind alle übrigen Kanten



## Graphalgorithmen

### *Klassifikation von Kanten*

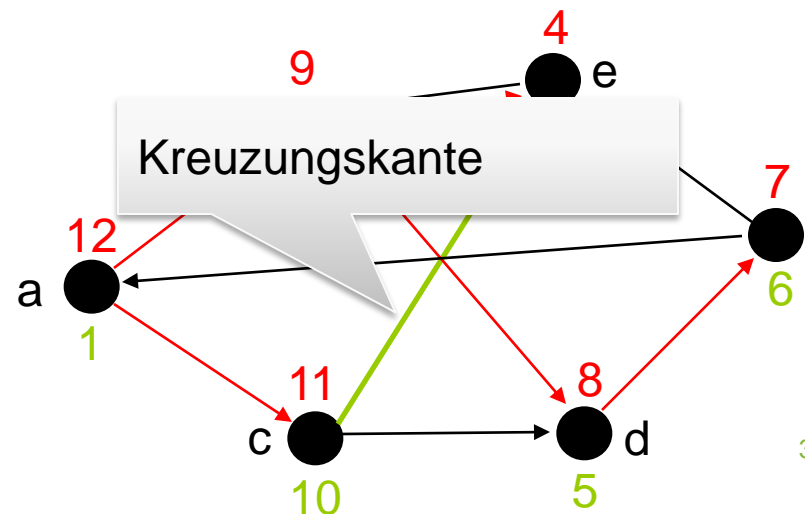
- **Baumkanten** sind Kanten des DFS-Walds  $G$
- **Rückwärtskanten** sind Kanten  $(u,v)$ , die Knoten  $u$  mit Vorgängern von  $u$  im DFS-Baum verbinden
- **Vorwärtskanten** sind die nicht-Baum Kanten  $(u,v)$ , die  $u$  mit einem Nachfolger  $v$  in einem DFS-Baum verbinden
- **Kreuzungskanten** sind alle übrigen Kanten



## Graphalgorithmen

### Klassifikation von Kanten

- **Baumkanten** sind Kanten des DFS-Walds  $G$
- **Rückwärtskanten** sind Kanten  $(u,v)$ , die Knoten  $u$  mit Vorgängern von  $u$  im DFS-Baum verbinden
- **Vorwärtskanten** sind die nicht-Baum Kanten  $(u,v)$ , die  $u$  mit einem Nachfolger  $v$  in einem DFS-Baum verbinden
- **Kreuzungskanten** sind alle übrigen Kanten



## Graphalgorithmen

### *Problem*

- Eingabe: Schaltkreis (ohne Zyklen) mit NOT, AND, OR, XOR Gattern,  $n$  Eingänge und  $m$  Ausgänge sowie eine Belegung der Eingänge durch boolesche Werte
- Ausgabe: Die Ausgabewerte des Schaltkreises bei dieser Belegung

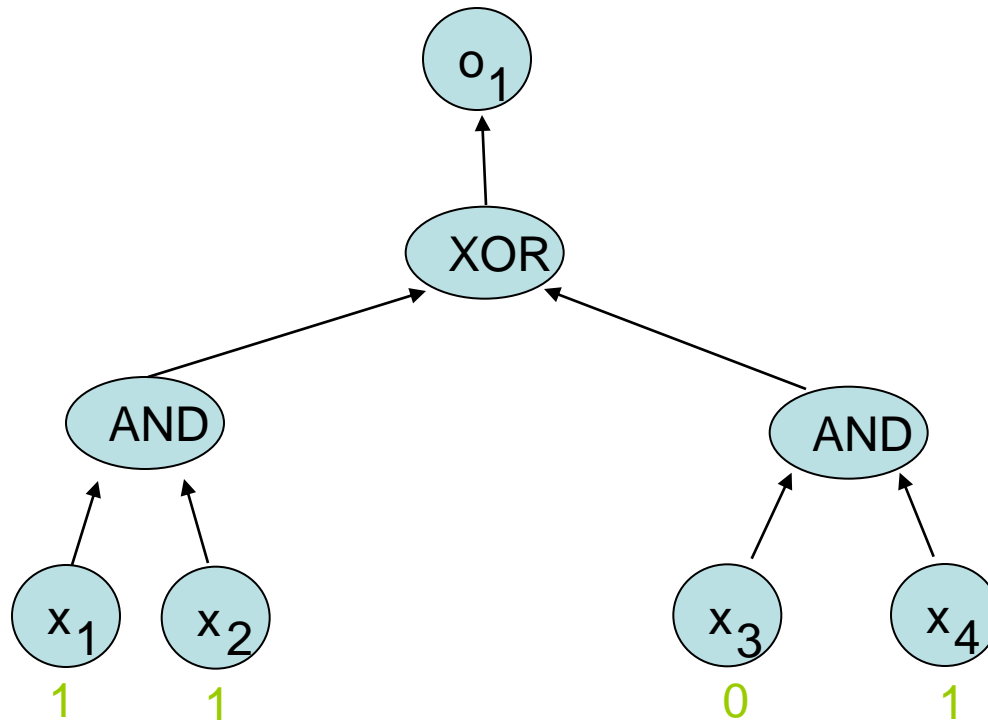
### *Bemerkung*

- Wir nehmen an, dass der Schaltkreis als gerichteter Graph gegeben ist. Jeder Knoten entspricht einem Gatter.

## Graphalgorithmen (siehe Vollversion)

### Beispiel:

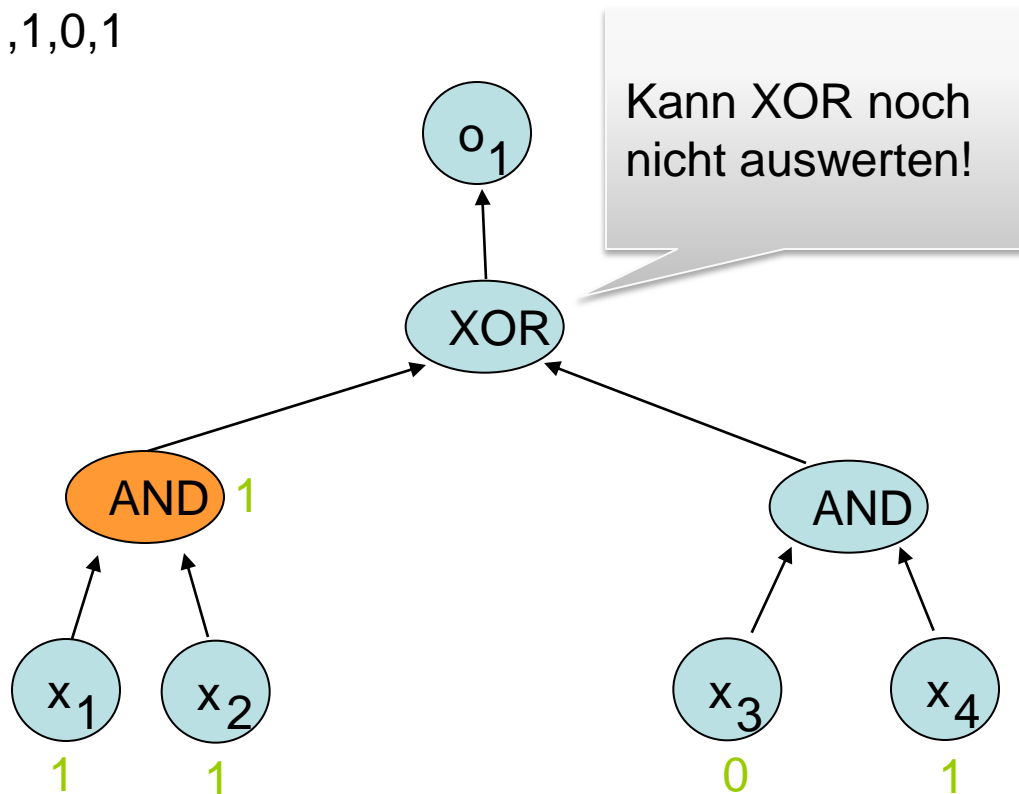
- Eingabe: 1,1,0,1



## Graphalgorithmen (siehe Vollversion)

### Beispiel:

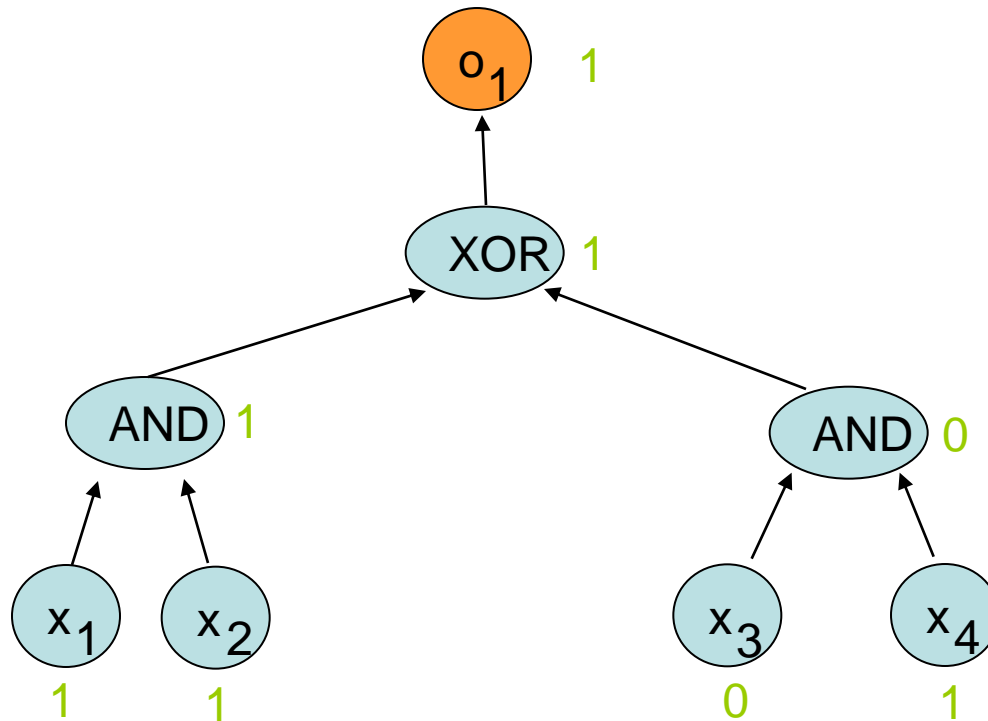
- Eingabe: 1,1,0,1



## Graphalgorithmen (siehe Vollversion)

### Beispiel:

- Eingabe: 1,1,0,1



## Graphalgorithmen

### *Frage*

- Wie kann man eine Reihenfolge der Knoten berechnen, so dass man an jedem Knoten sofort den Wert des Gatters ausrechnen kann?

### *Eine Möglichkeit*

- Topologisches Sortieren
- Sortierung eines gerichteten, kreisfreien Graphen, so dass für jede Kante  $(u,v)$   $u$  in der Sortierung vor  $v$  steht („ $u$  wird vor  $v$  ausgewertet“)

## Graphalgorithmen

### Topsort(G)

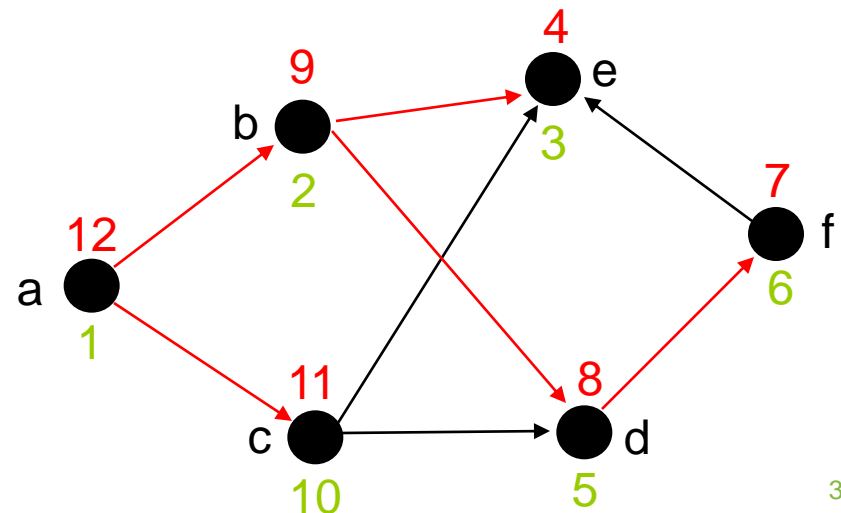
1. Rufe DFS(G) auf, um Zeitstempel  $f[v]$  für jeden Knoten  $v$  zu berechnen
2. Sobald ein Knoten abgearbeitet ist, füge ihn zu Beginn einer Liste  $L$  ein (absteigende Sortierung nach  $f[v]$ -Werten)
3. **return**  $L$

### Sortierung im Beispiel

- a, c, b, d, f, e

### Laufzeit

- $O(|V|+|E|)$



## Graphalgorithmen

### *Lemma 70*

Ein gerichteter Graph  $G$  ist azyklisch, gdw. Eine Tiefensuche in  $G$  keine Rückwärtskanten hat.

### *Beweis*

„ $\Rightarrow$ “: Annahme: Es gibt Rückwärtskante  $(u,v)$ . Dann ist  $v$  Vorgänger von  $u$  im DFS Wald. Es gibt also einen Weg von  $v$  nach  $u$  und die Rückwärtskante schließt den Kreis.

„ $\Leftarrow$ “: Annahme:  $G$  enthält Kreis  $C$ . Sei  $v$  der erste Knoten aus  $c$ , der entdeckt wurde und sei  $(u,v)$  die vorhergehende Kante im Kreis  $C$ . Zum Zeitpunkt  $d[v]$  gibt es einen weißen Pfad von  $v$  nach  $u$ . Nach Satz 67 wird  $u$  Nachfolger von  $v$  und  $(u,v)$  somit Rückwärtskante.

## Graphalgorithmen

### Satz 71

TopSort(G) berechnet eine topologische Sortierung eines gerichteten azyklischen Graphs.

### Beweis

- Annahme: Wir führen DFS auf gerichtetem Graph aus, um die Abarbeitungszeitpunkte zu bestimmen.
- Z.Z.: Für jedes Paar von Knoten  $u, v \in V$  mit  $u \neq v$  gilt: Gibt es Kante von  $u$  nach  $v$ , so ist  $f[v] < f[u]$ .
- Betrachte Zeitpunkt, wenn eine beliebige Kante  $(u, v)$  durch Tiefensuche entdeckt wird. Dann kann  $v$  nicht grau sein, denn dann wäre  $(u, v)$  Rückwärtskante und nach Lemma 70  $G$  nicht azyklisch.
- Daher ist  $v$  entweder weiß oder schwarz.
- Ist  $v$  weiß, so wird  $v$  Nachfolger von  $u$  und es gilt  $f[v] < f[u]$
- Ist  $v$  schwarz, dann gilt erst recht  $f[v] < f[u]$ . Damit folgt der Satz.