Kap. 6.3: Traversieren von Graphen
Kap. 6.4: Elementare Graphalgorithmen

Carsten Gutwenger

Lehrstuhl für Algorithm Engineering, LS11

Fakultät für Informatik, TU Dortmund

18. VO DAP2 SS 2009 25. Juni 2009

Motivation

Heute braucht es keine Motivation, denn:

Spielereien

DFS

mit

Überblick

- Traversieren von Graphen:
  - Breitensuche (BFS)
  - Tiefensuche (DFS)

- Elementare Graphenalgorithmen:
  - Zusammenhangskomponenten
  - Kreise in Graphen
  - Topologisches Sortieren

Kap. 6.3.2 Tiefensuche

Tiefensuche (DFS)

engl.: Depth-first-search, DFS

Idee: besuche die Knoten rekursiv: wenn v zum ersten Mal gesehen wird, markiere ihn als gesehen und erforsche den Graphen von v aus weiter!

- Im Gegensatz zur Breitensuche, wird hier der Graph erst einmal in seiner Tiefe durchdrungen.
- Bei Breitensuche werden erst alle gesehenen Knoten bearbeitet, bevor die neuen bearbeitet werden (Queue). Hier wir immer erst das Neue erledigt.

Tiefensuche DFS

Methode: DFS-VISIT(v):

- Durchlaufe alle Nachbarn von v
- Für jeden neu entdeckten (nicht markierten) Knoten w:
  - Markiere w;
  - rufe DFS-VISIT(w) auf.

- Der folgende Algorithmus DFS(G) durchwandert alle Knoten des Graphen.
- Im Feld π[w] wird jeweils der Vorgänger von w (also v) gespeichert.

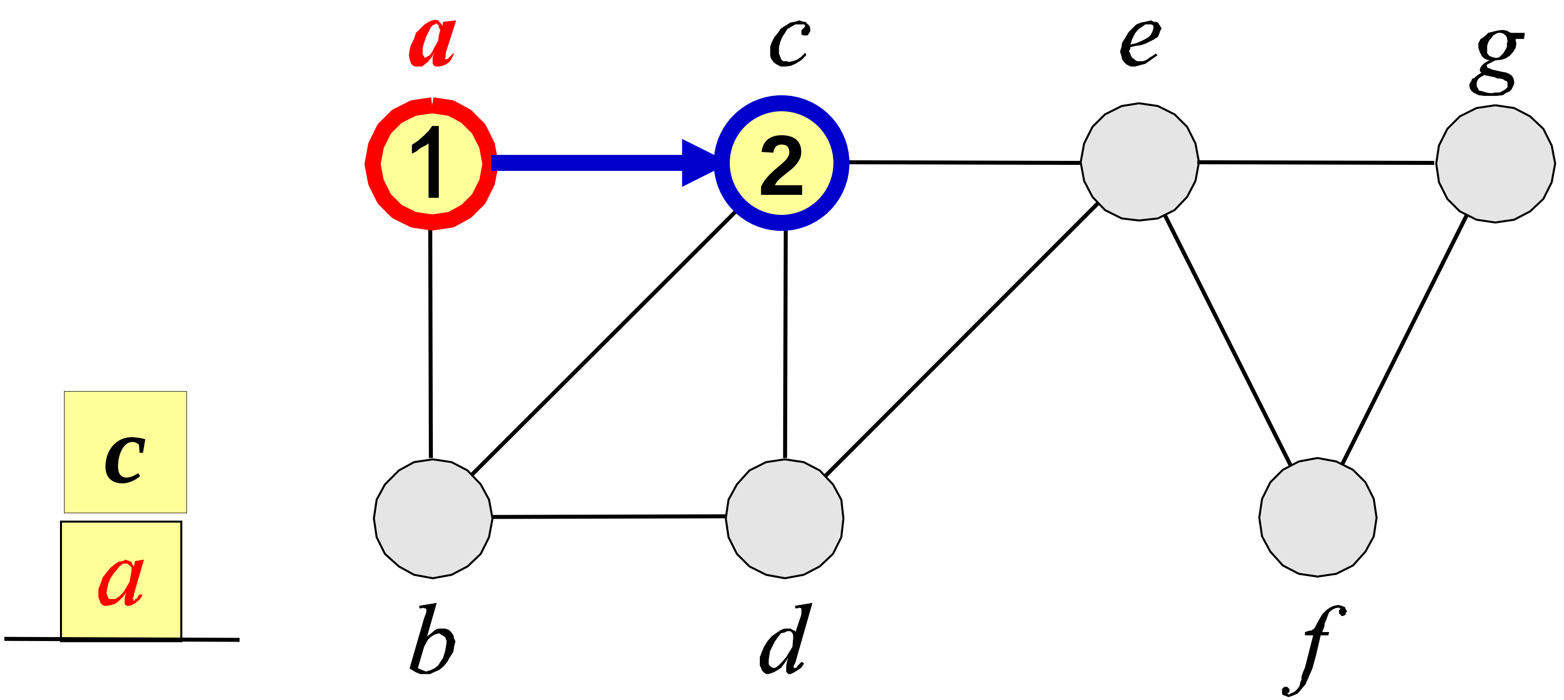
Algorithmus DFS(G)

Sei $G=(V,E)$ ungerichteter Graph, v,w: Knoten

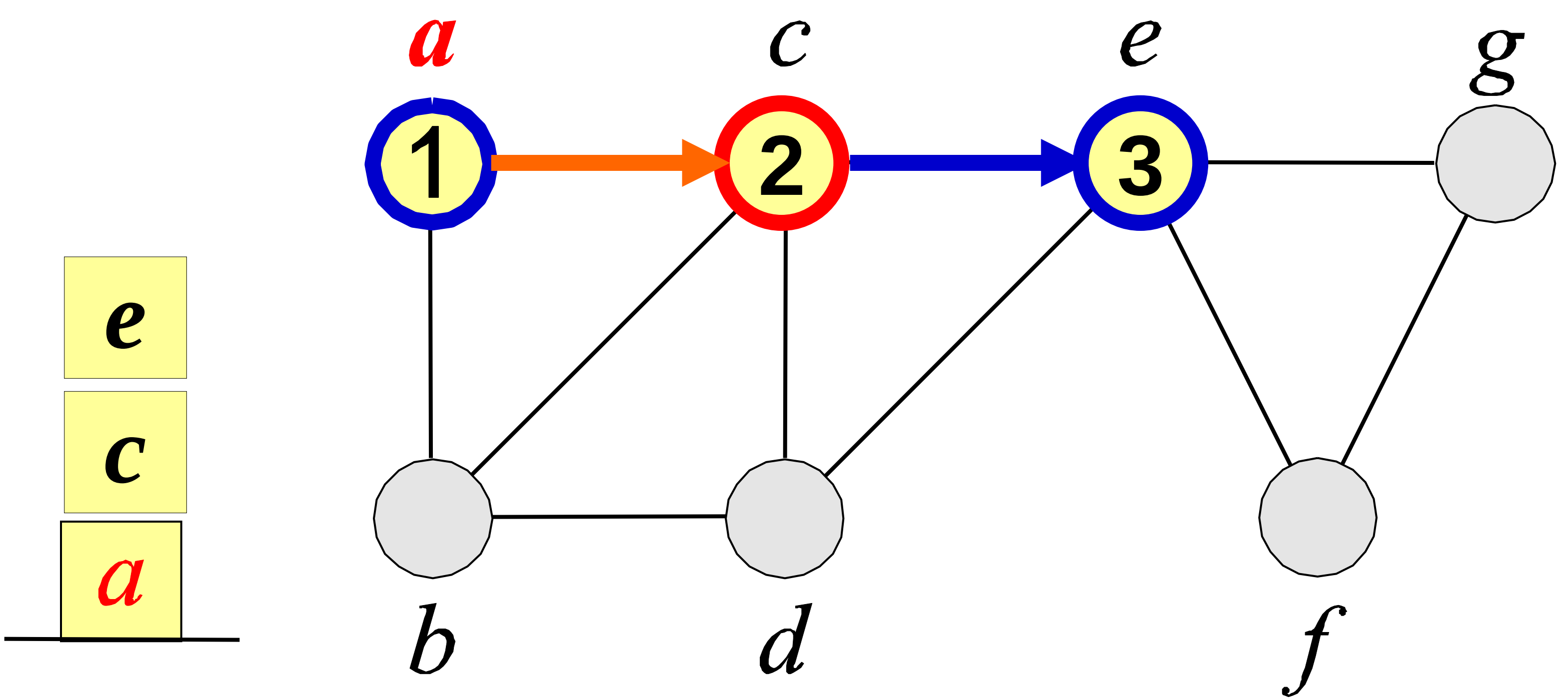
```
for all v∈V do { marked[v]:=false; π[v]:=nil}
for all v∈V do
    if not marked[v] then DFS-VISIT(v)

procedur DFS-VISIT(Node v)
    marked[v]:=true
    for all w∈N(v) do
        if not marked[w] then {
            π[w]:=v
            DFS-VISIT(w)
        }
```

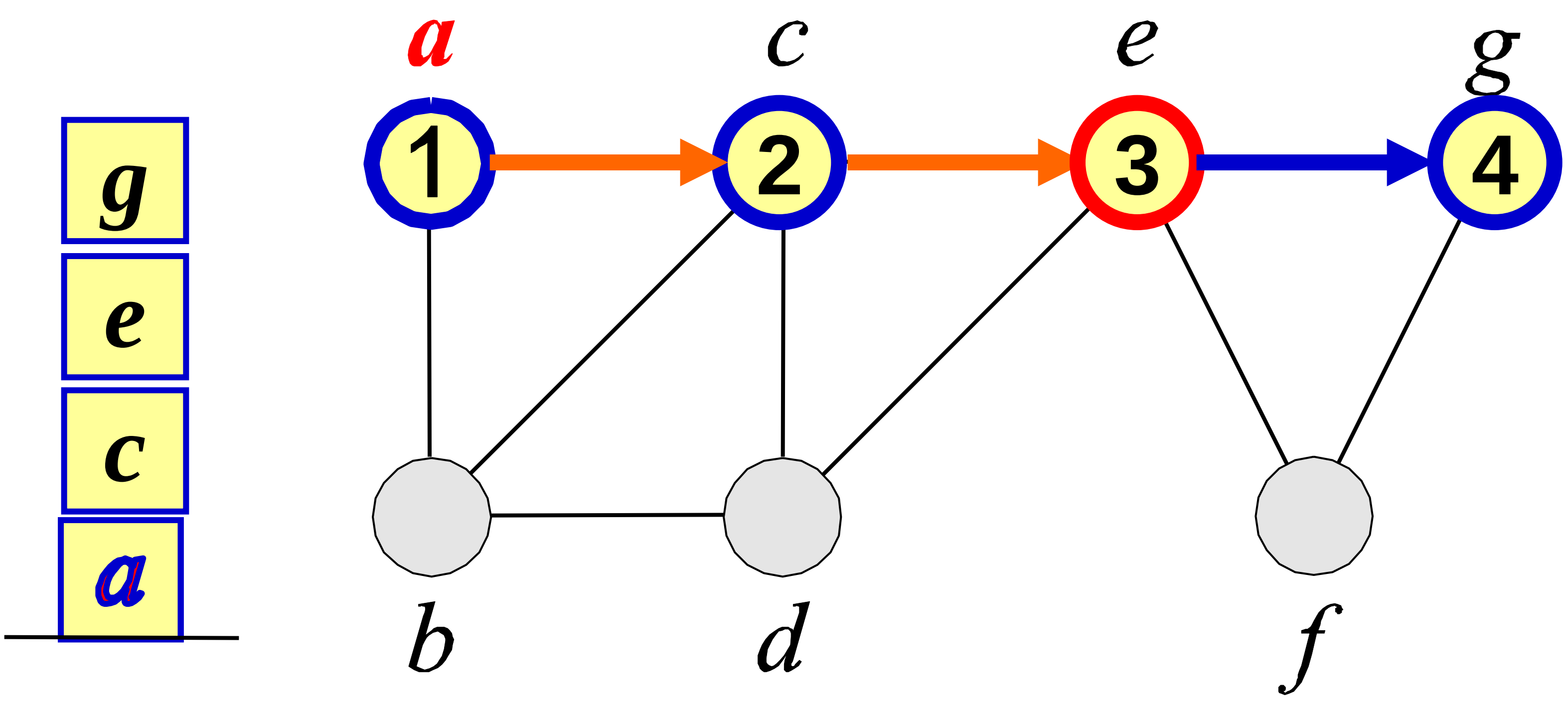
Beispiel für DFS



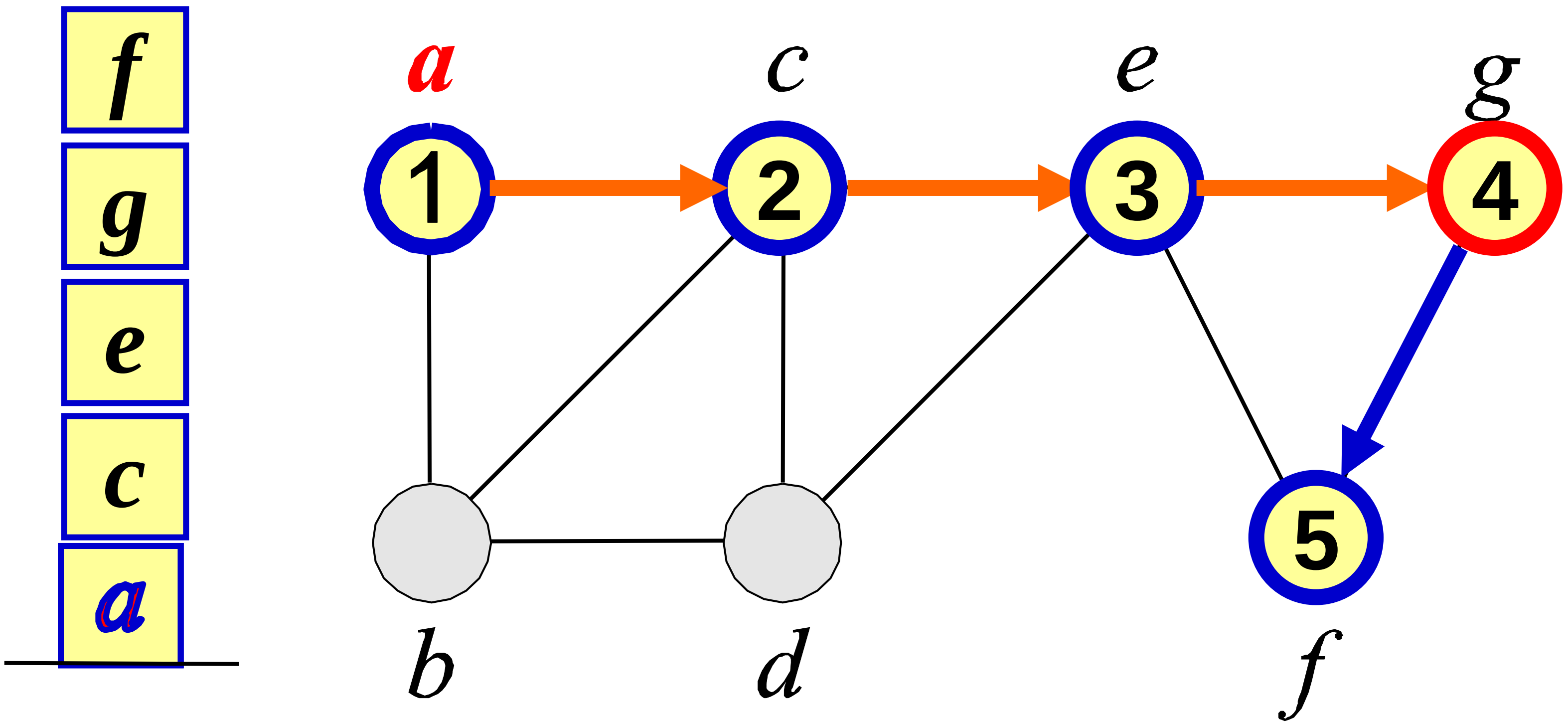
Beispiel für DFS



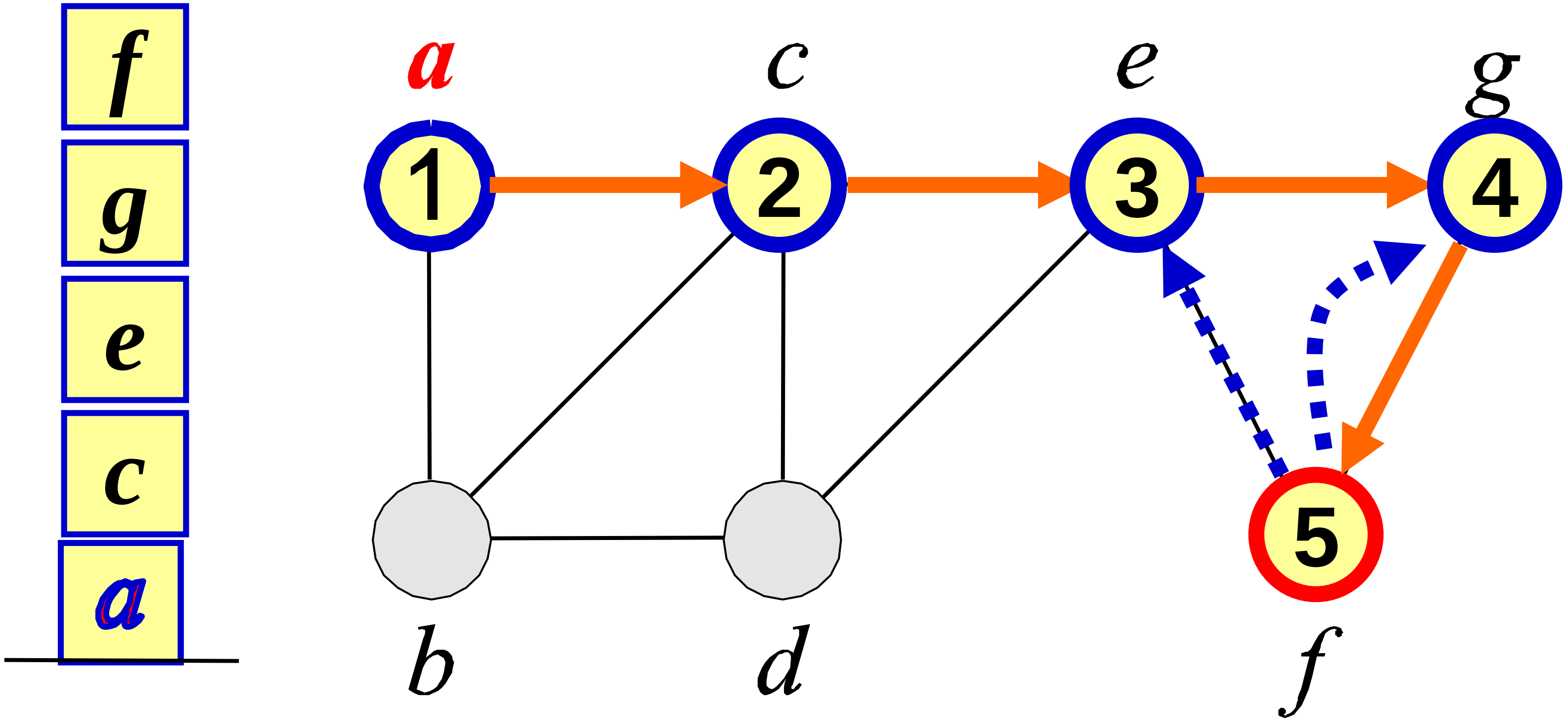
Beispiel für DFS



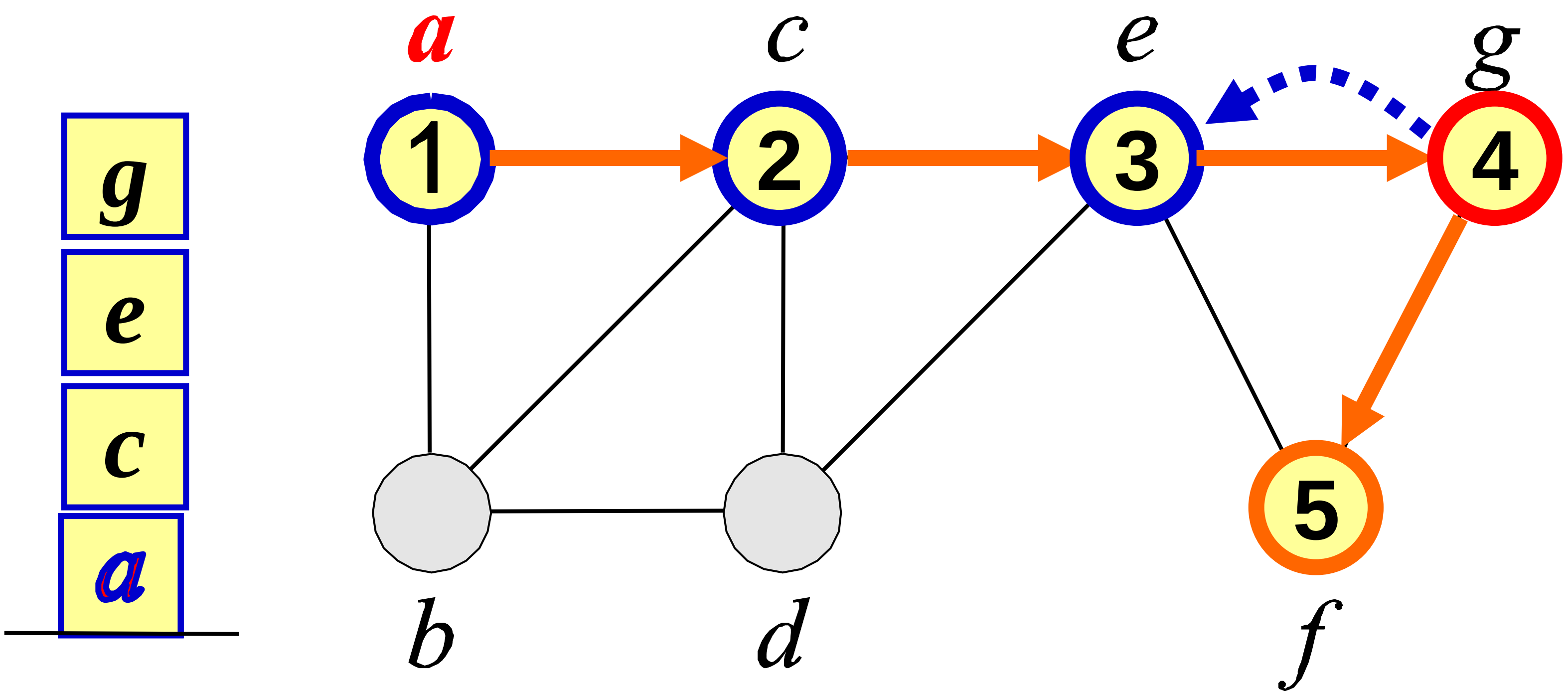
Beispiel für DFS



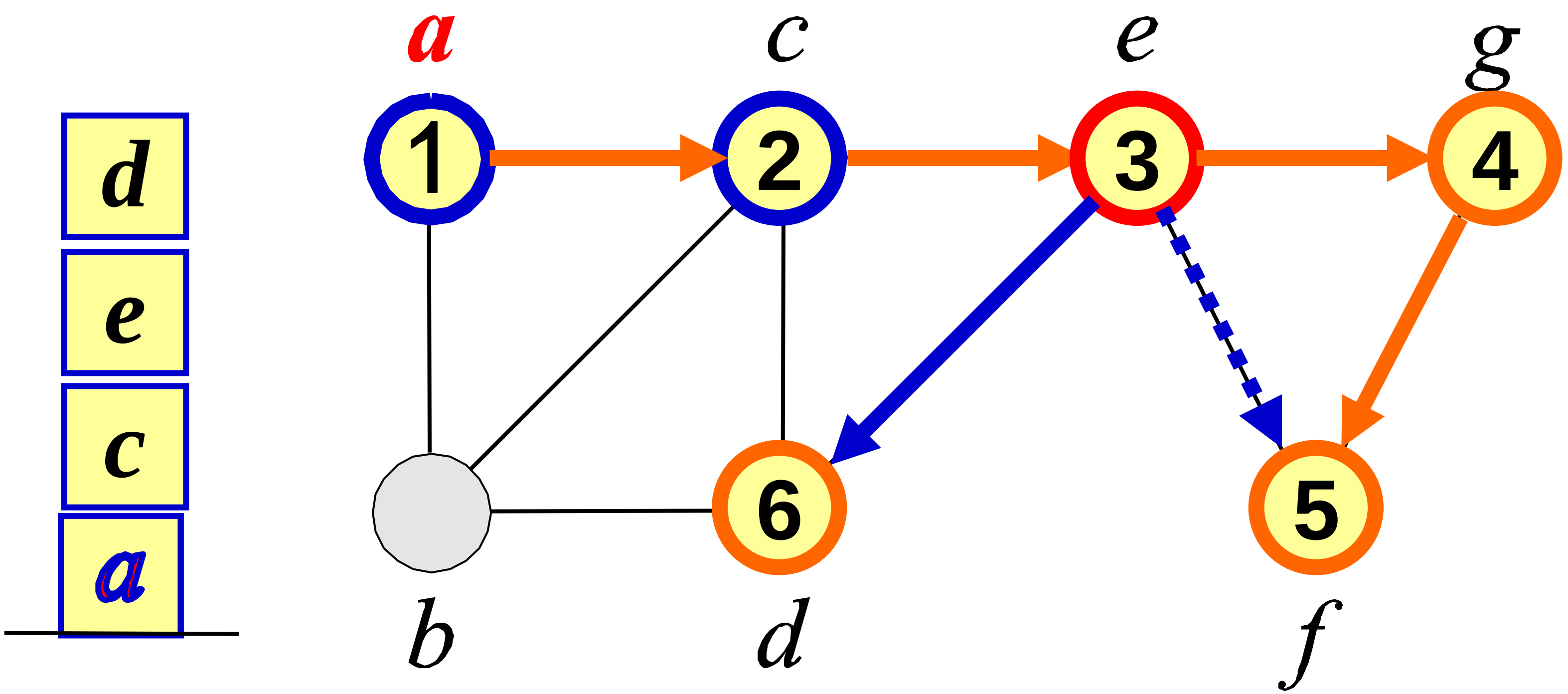
Beispiel für DFS



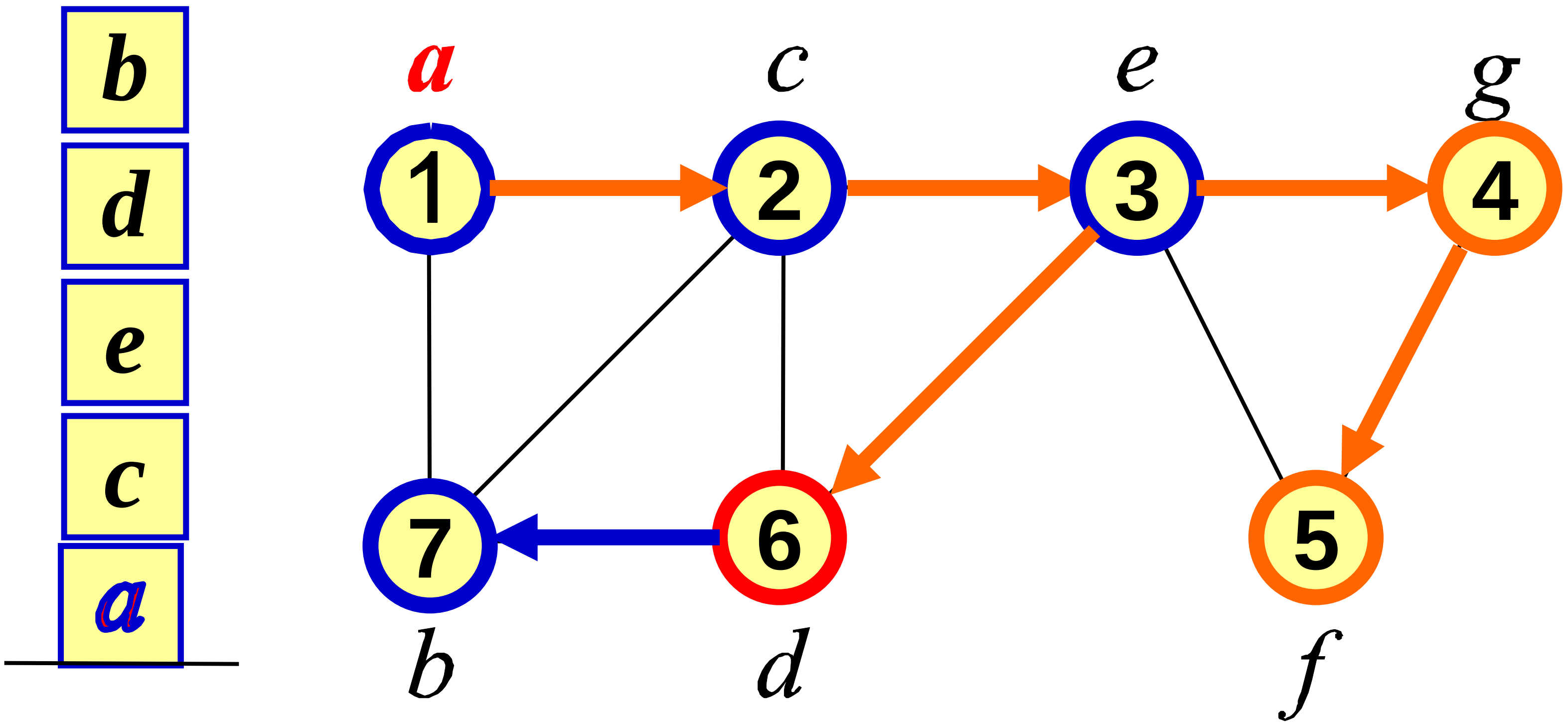
Beispiel für DFS



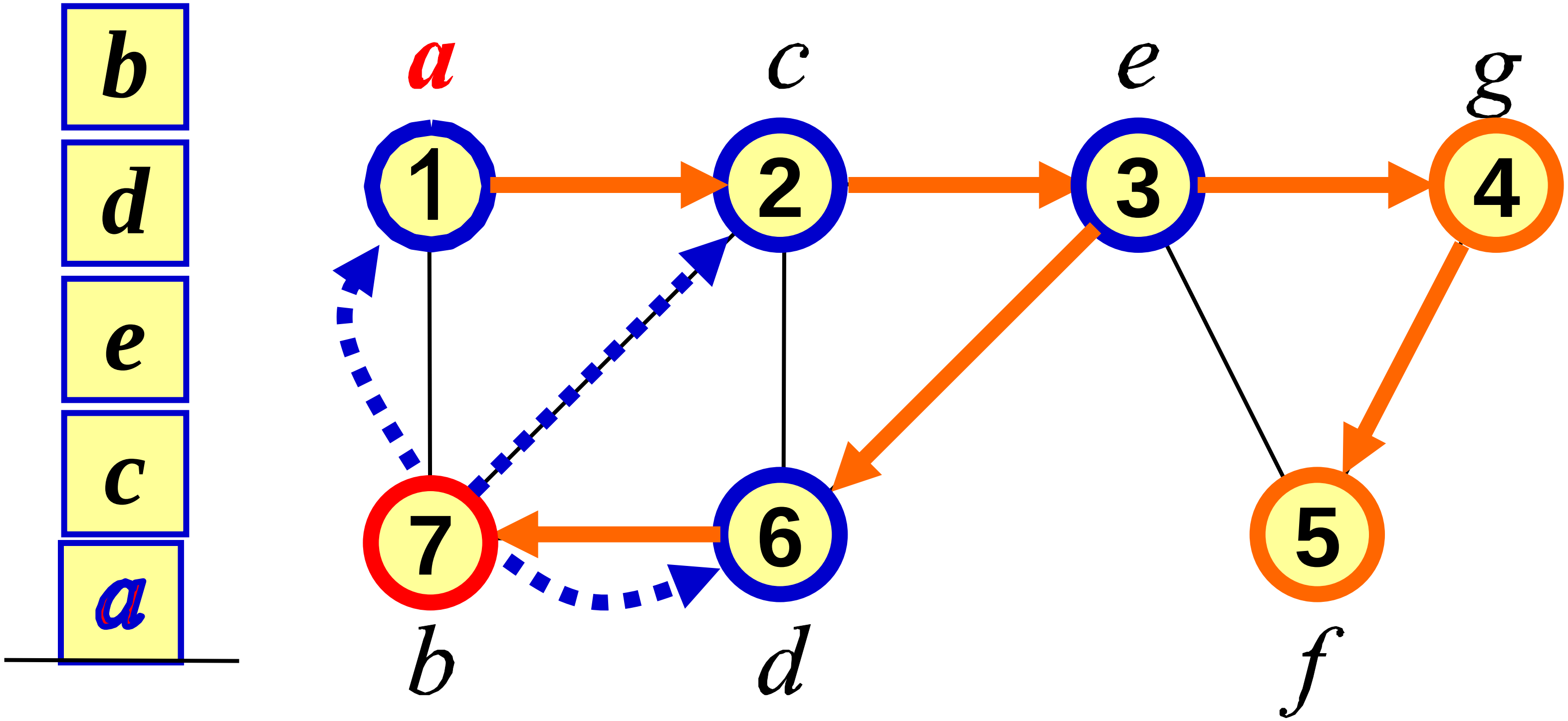
Beispiel für DFS



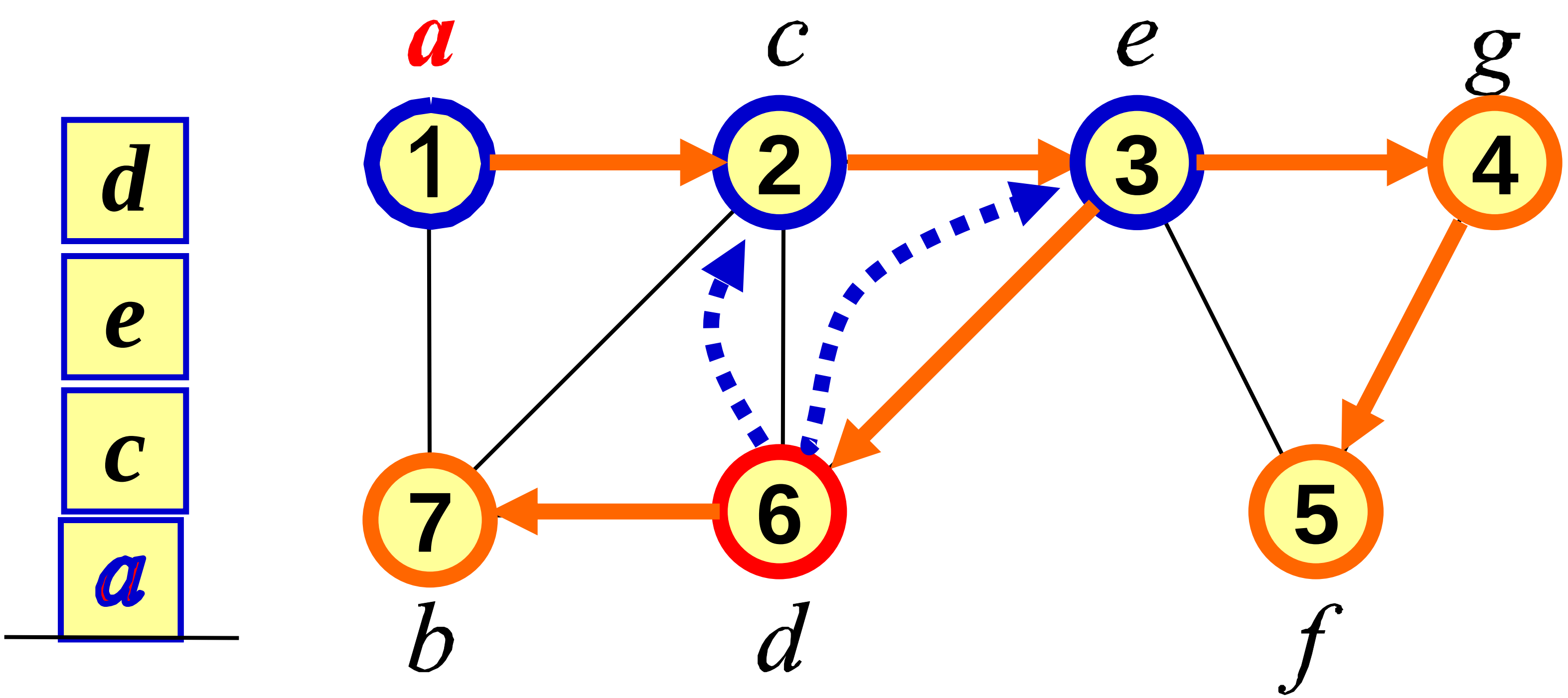
Beispiel für DFS



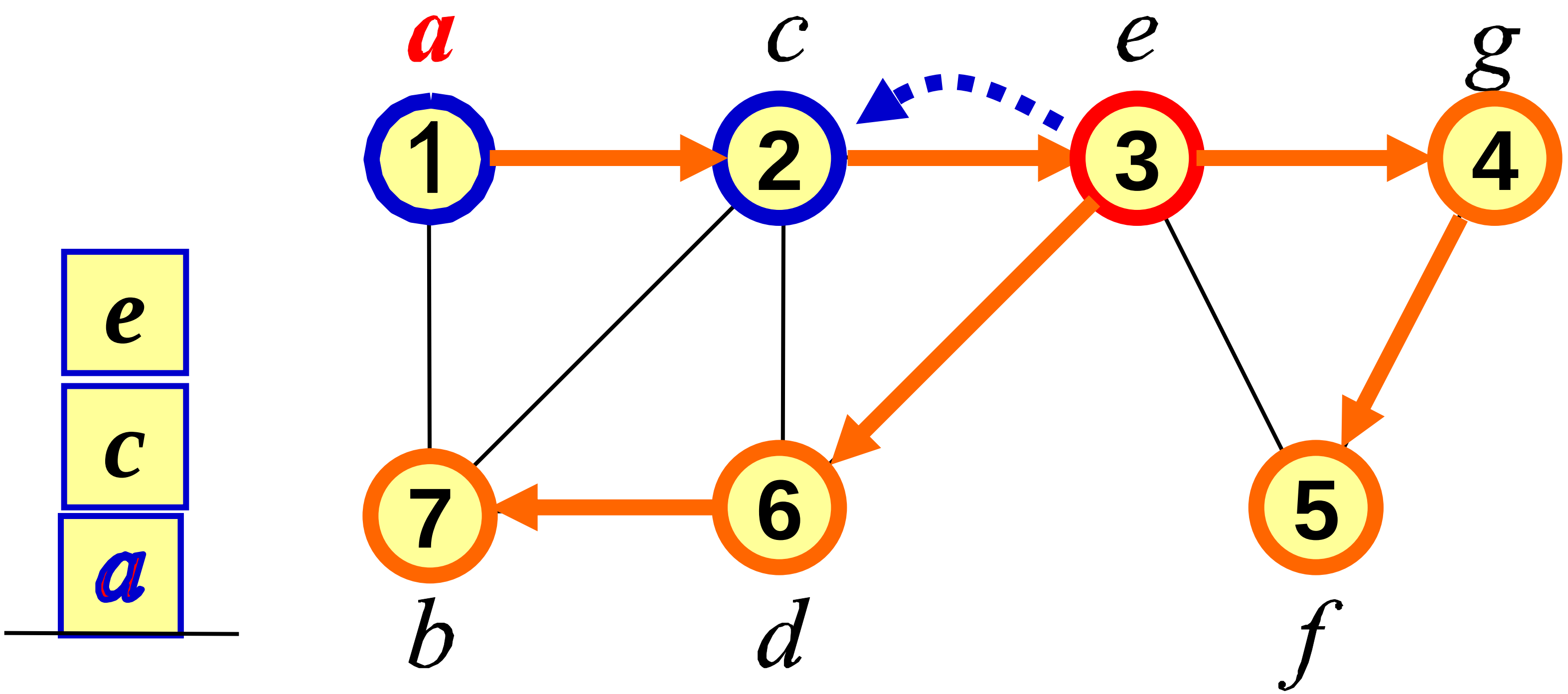
Beispiel für DFS



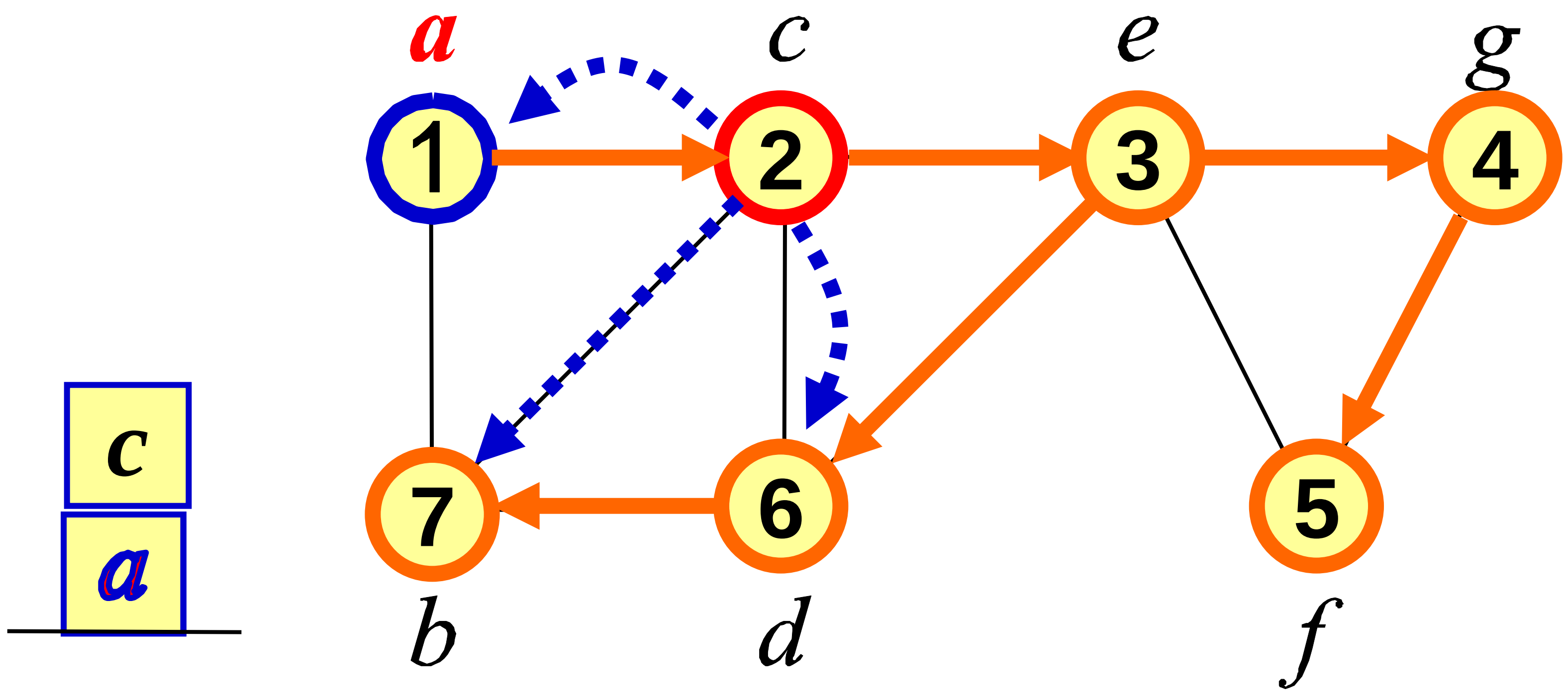
Beispiel für DFS



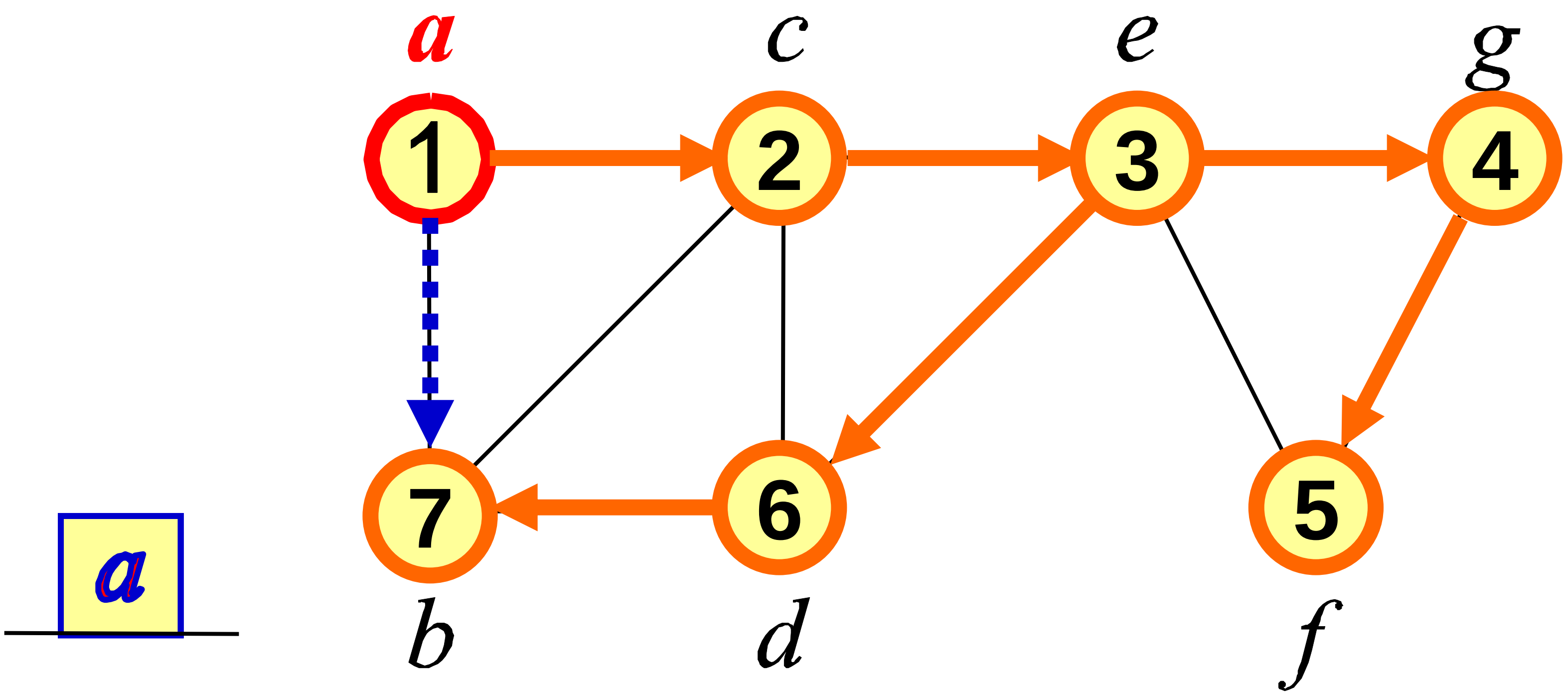
Beispiel für DFS



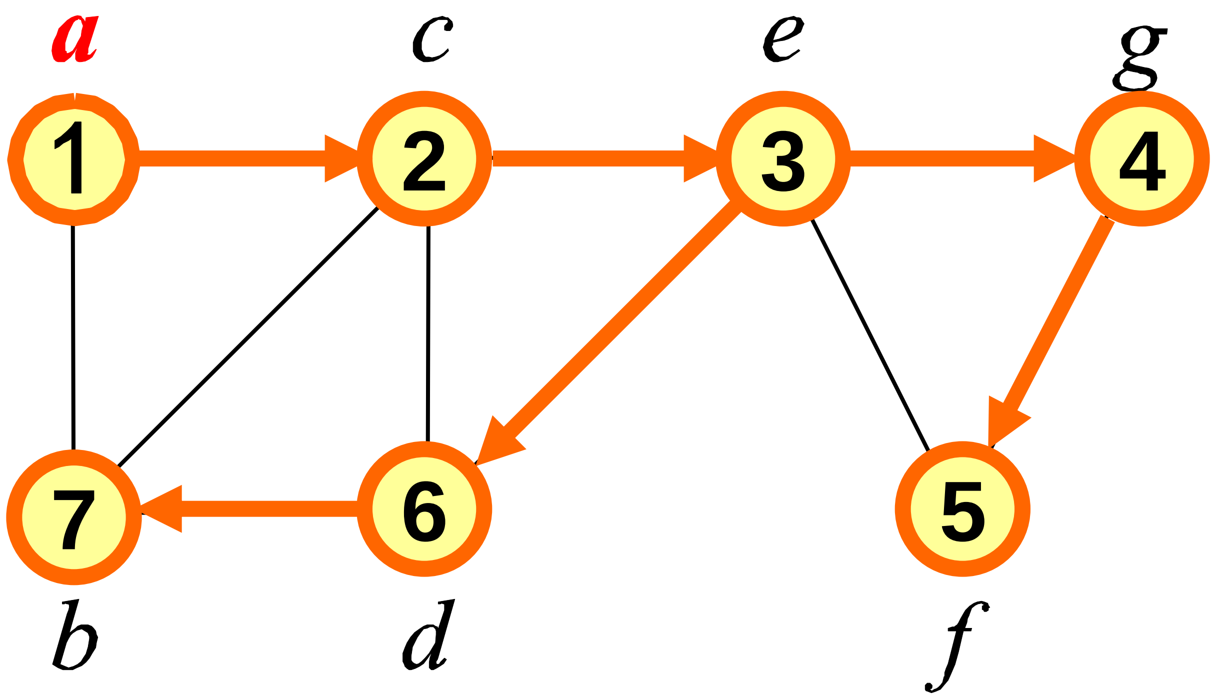
Beispiel für DFS



Beispiel für DFS



Beispiel für DFS



Die orangen Kanten (diejenigen Kanten (u,v), die zum ersten Mal v besuchen) bilden einen Baum: den DFS-Baum (G zusammenhängend, sonst Wald)

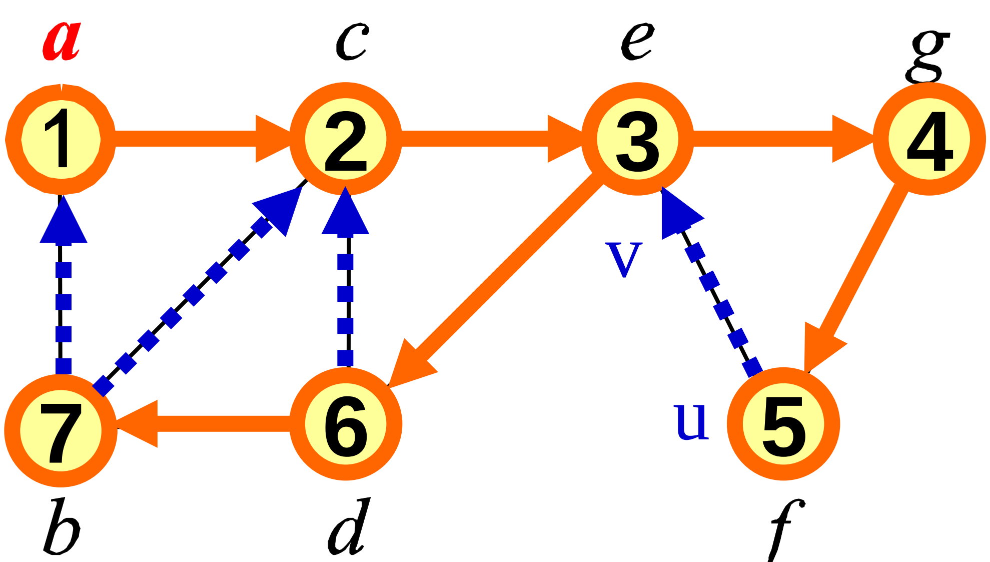
Sei G zusammenhängend

DFS-Baum

DFS-Baum besteht aus

- Baumkanten (tree edges, T-Kanten): $(\pi(v),v)$
- Rückwärtskanten (back edges, B-Kanten): s.u.

- DFS teilt die Kanten in T-Kanten und B-Kanten
- Für jede B-Kante (u,v) gilt: Es gibt genau einen Weg von v nach u mit T-Kanten im DFS-Baum

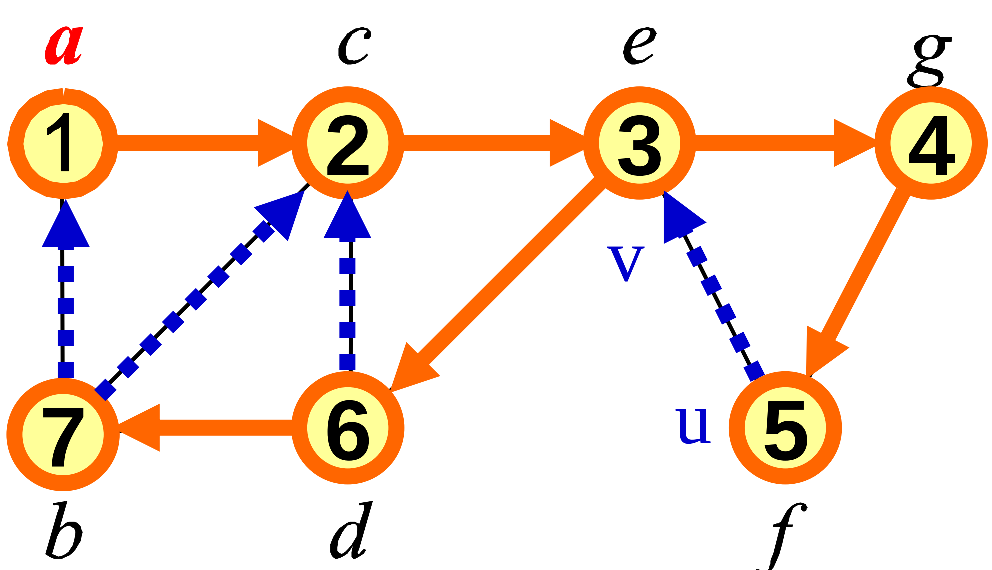


Sei G zusammenhängend

DFS-Baum

- Jeder Knoten erhält eine eindeutige DFS-Nummer.
- Für die Rückwärtskanten (u,v) gilt: DFSNUM(u) > DFSNUM(v).

- DFS teilt die Kanten in T-Kanten und B-Kanten
- Für jede B-Kante (u,v) gilt: Es gibt genau einen Weg von v nach u mit T-Kanten im DFS-Baum



Analyse von DFS

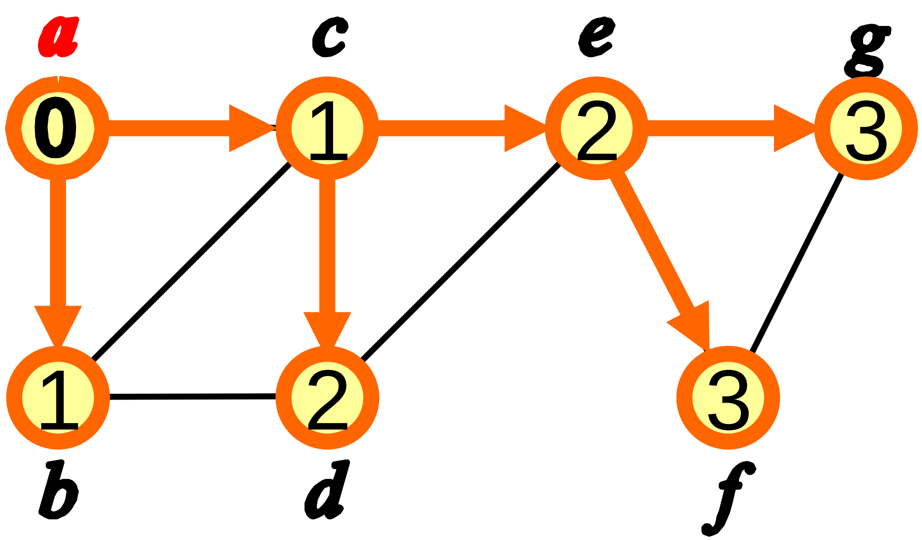
- Initialisierung: $\Theta(|V|)$
- Die **for-all**-Schleife ohne Aufrufe von DFS-VISIT(): $O(|V|)$.
- DFS-VISIT() wird für jeden Knoten genau einmal aufgerufen.
- Jeder Aufruf von DFS-VISIT(v) (ohne Kosten der rekursiven Aufrufe) benötigt $\Theta(d(v))$ Zeit.

- Gesamtaufwand: $\Theta(|V|)+\sum_{v\in V} \Theta(d(v))= \Theta(|V|+|E|)$

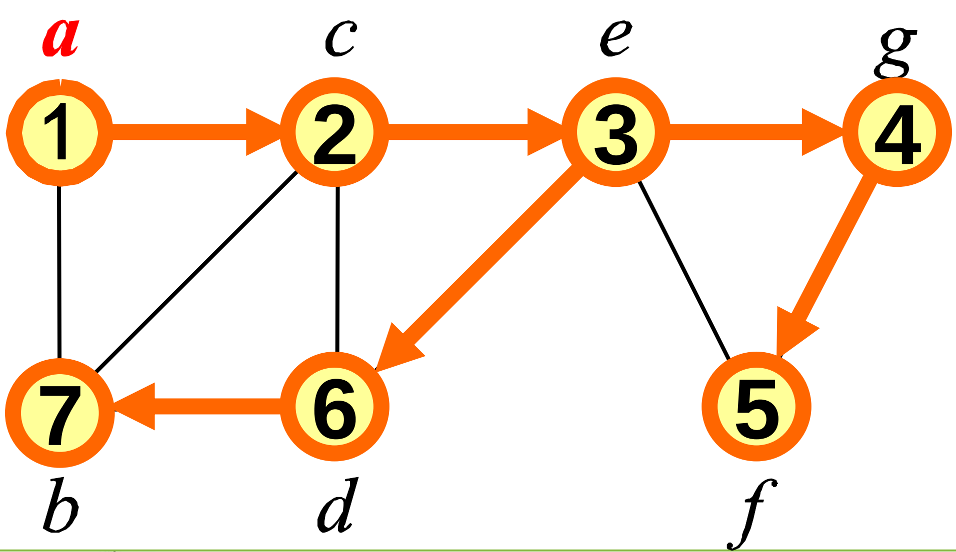
BFS-Baum vs. DFS-Baum

BFS-Baum

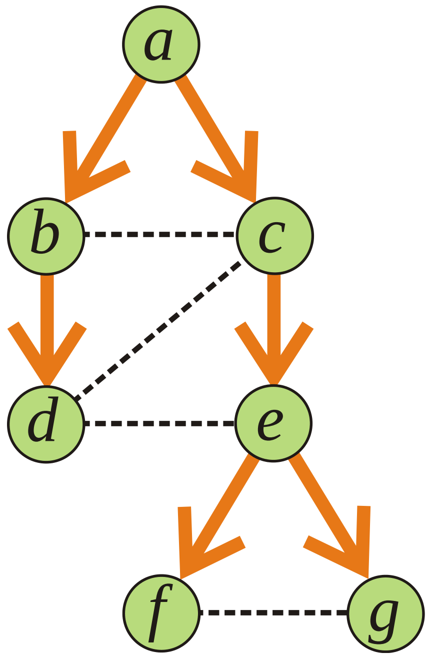
Höhe eindeutig



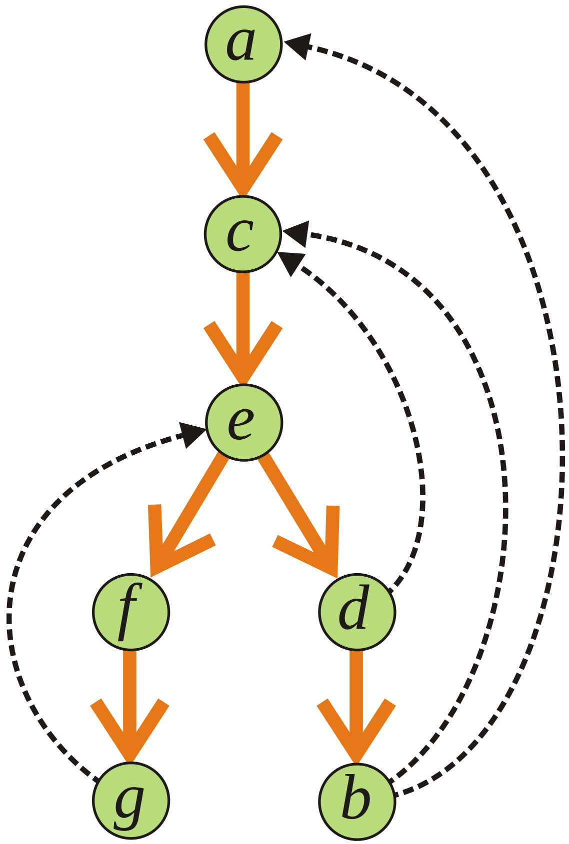
DFS-Baum



BFS & DFS



BFS-Baum



DFS-Baum

Kap. 6.4 Elementare Graphenalgorithmen

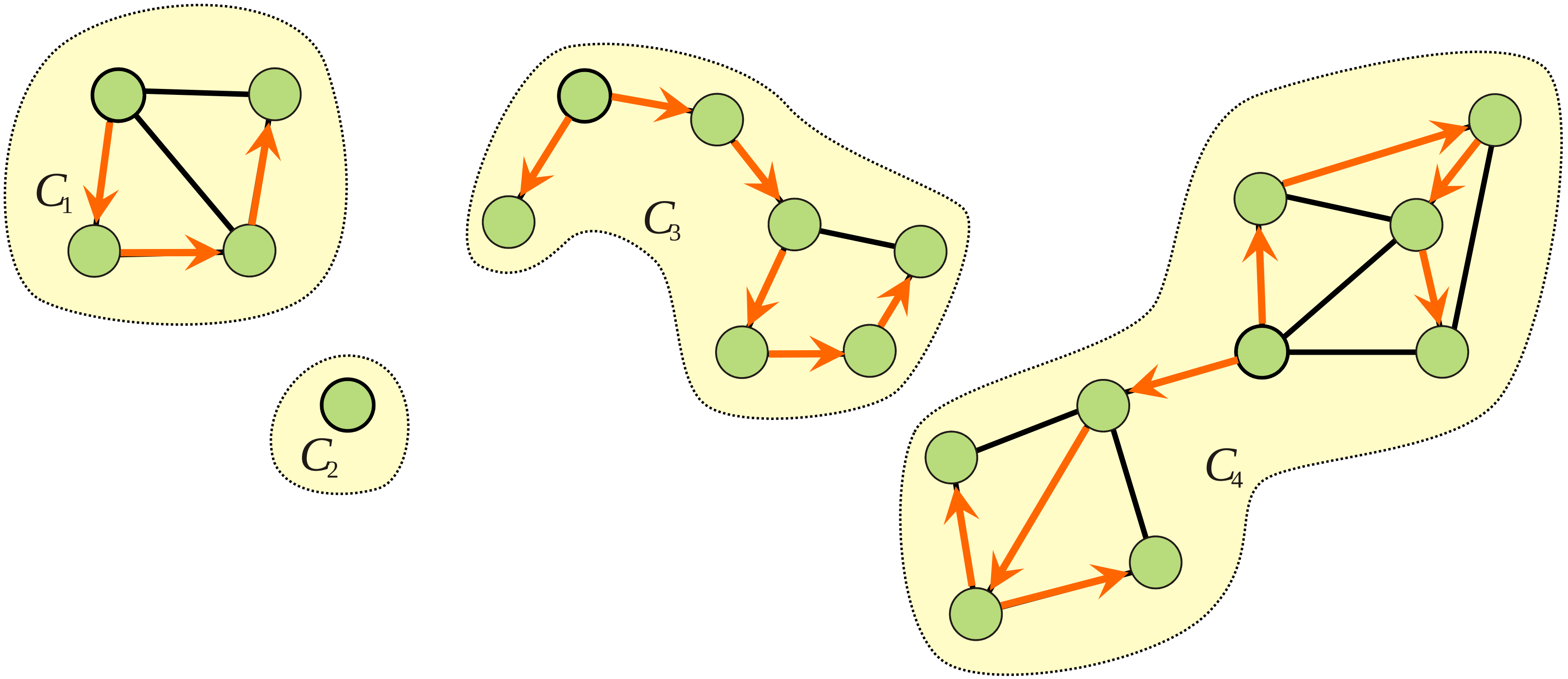
Kap. 6.4.1 Komponenten eines Graphen

Zusammenhangskomponenten

Definitionen (Zusammenhang)

- G heißt zusammenhängend (connected), wenn $|V| \geqq 1$ und für jedes Paar u,v von Knoten ein Weg von u nach v in G existiert.
- Ein Graph $G´=(V´,E´)$ mit $V´ \subseteq V$ und $E´ \subseteq E$ ist ein Untergraph (Teilgraph, subgraph) von G. Wir schreiben $G´ \subseteq G$.
- Eine (Zusammenhangs-) Komponente (component) von G ist ein maximaler zusammenhängender Untergraph U von G, d.h. es gibt keinen anderen zusammenhängenden Untergraphen $U´$ mit $U \subseteq U´$.

Komponenten eines Graphen



→ T-Kanten des DFS-Waldes

Bestimmen der Zusammenhangskomponenten

Idee:

- Beginne an einem Knoten s und bestimme alle von s aus erreichbaren Knoten $\rightarrow$ Komponente 1 (Markierung mit 1)
- Beginne nun bei einem noch unmarkierten Knoten v ($\mathrm{marked}==0$) und bestimme alle von v aus erreichbaren Knoten; markiere dabei mit 2 $\rightarrow$ Komponente 2
- …u.s.w. bis alle Knoten markiert sind

Algorithmus COMPONENTS(G)

Sei $G=(V,E)$ ungerichteter Graph, v,w: Knoten

```
for all v∈V do { marked[v]:=0}
numComps:=0
for all v∈V do {
    if marked[v]=0 then {
        numComps:=numComps+1
        DFS-COMP(v)
} }
Procedure DFS-COMP(Node v)
    marked[v]:=numComps
    for all w∈N(v) do
        if marked[w]=0 then
            DFS-COMP(w)
```

Bestimmen der Zusammenhangskomponenten

Theorem: Der Algorithmus COMPONENTS(G) bestimmt die Zusammenhangskomponenten eines Graphen $G=(V,E)$ in Zeit $\Theta(|V|+|E|)$.

Kap. 6.4.2 Kreise in Graphen

- Aufgabe: Gegeben ist ein ungerichteter Graph $G=(V,E)$. Enthält G einen Kreis?

- Idee: mittels DFS

Kreise und DFS

- **Lemma:** Sei G ein ungerichteter Graph und F ein beliebiger DFS-Wald für G. Dann ist G genau dann kreisfrei, wenn G keine Rückwärtskante (B-Kante) bezüglich F enthält.

Beweis:

- 1. Fall: F enthält keine B-Kante: dann besteht F nur aus T-Kanten $\rightarrow$ F enthält alle Kanten aus $G \rightarrow G$ kreisfrei
- 2. Fall: Sei (u,v) B-Kante bzgl. $F \rightarrow$ es existiert Weg $v,\ldots,u$ mit T-Kanten in $F \rightarrow$ Weg mit Kante (u,v) bildet Kreis in G.

Kreise und DFS

- **Methode:** Erweitere DFS um Speicherung der B-Kanten: sobald wir von v aus auf einen markierten Knoten u treffen, haben wir eine B-Kante gefunden, falls u nicht der direkte Vorgänger (Elter) von v ist.

Algorithmus ISACYCLIC(G) speichert dabei alle B-Kanten. Wenn man nur einen Kreis finden möchte, dann kann man abbrechen, sobald man eine B-Kante gefunden hat.

Algorithmus ISACYCLIC(G)

Sei G=(V,E) ungerichteter Graph

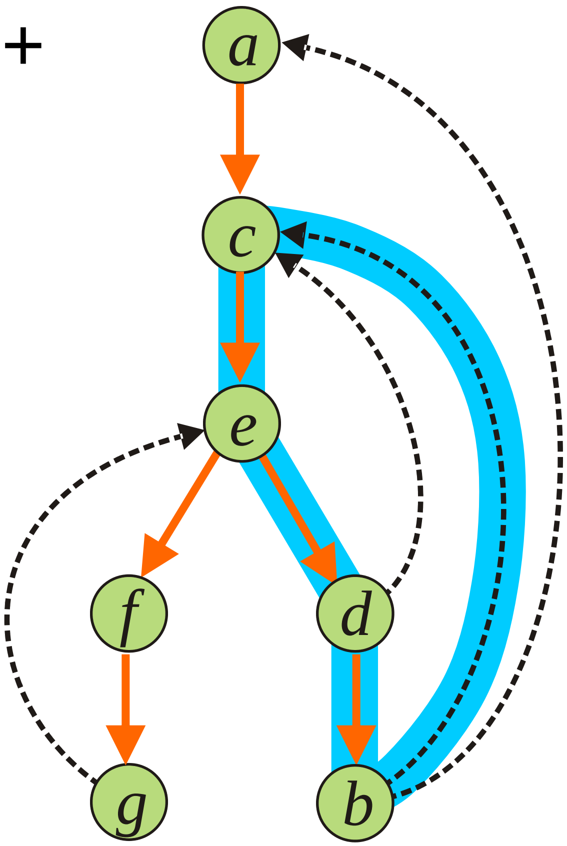
```
function ISACYCLIC(Graph G=(V,E)) : bool {
    B := ∅
    for all v∈V do { marked[v]:=false; π(v):=nil }
    for all v∈V do {
        if not marked[v] then
            DFS-ACYCLIC(v)
    }
    if B ≠ ∅ then return false else return true
}
```

Algorithmus ISACYCLIC(G) ff

```
procedure DFS-ACYCLIC(Node v) {
    marked[v] := true
    for all w ∈ N(v) do
        if not marked[w] then
            π[w] := v
            DFS-ACYCLIC(w)
        else if π [v] ≠ w then
            B := B ∪ (v,w)
    }
}
```

Welche Kreise werden gefunden?

Pfad von T-Kanten +
eine B-Kante



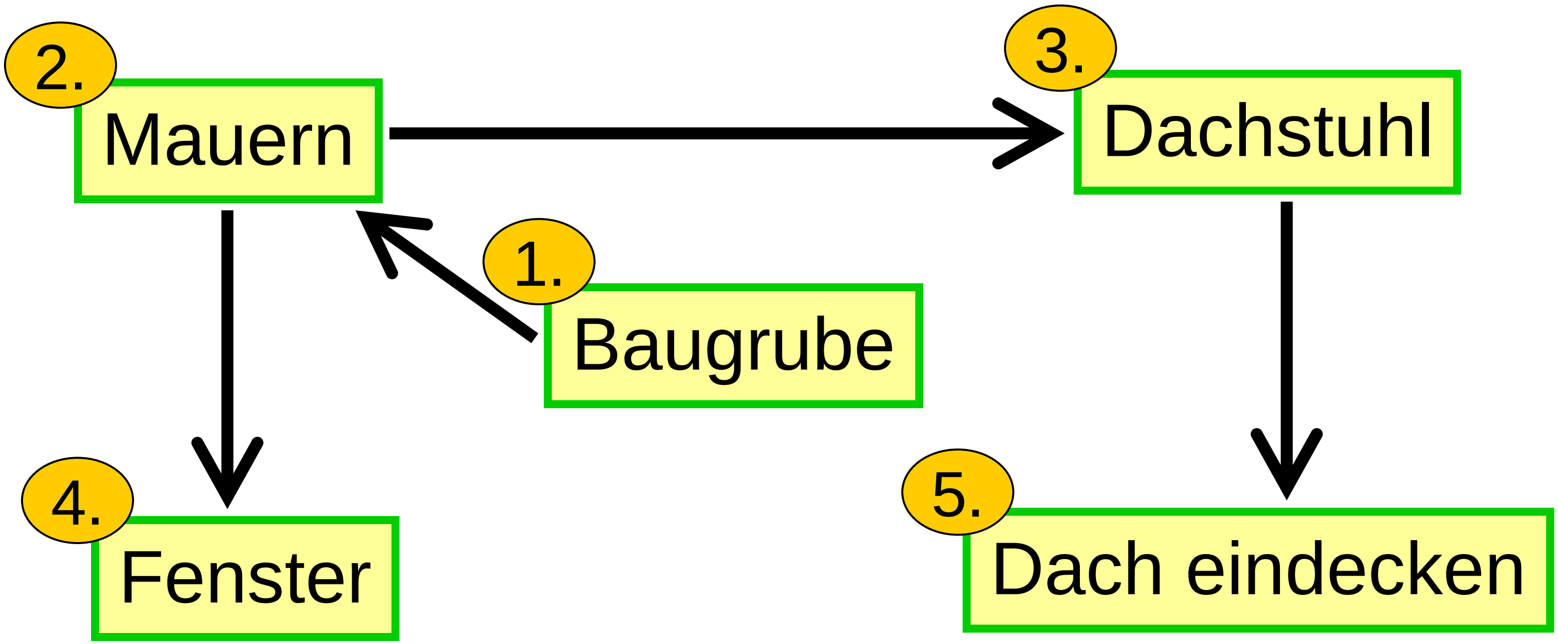
Test auf Kreisfreihet

Theorem: Die Funktion ISACYCLIC(G) testet einen ungerichteten Graphen $G=(V,E)$ auf Kreisfreiheit in Zeit $\Theta(|V|+|E|)$.

Kap. 6.4.3 Topologisches Sortieren

Achtung: in diesem Abschnitt
gerichtete Graphen!

Modellierung von Abhängigkeiten



$\rightarrow$ gerichteter Graph
Kante $(x,y) \Rightarrow$ Aufgabe x muss abgeschlossen werden, bevor mit y angefangen werden kann

DAG

Grundvoraussetzung:

Graph ist azyklisch!

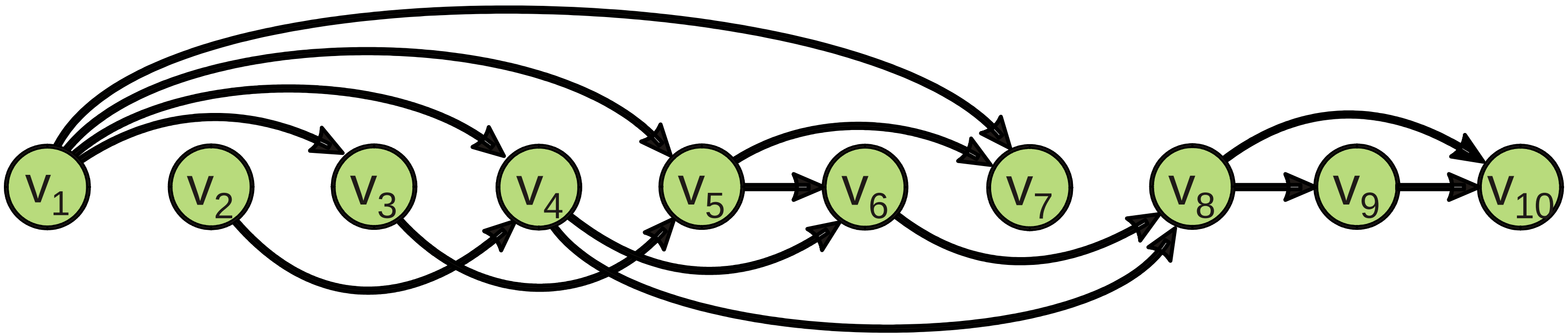
(also keine zyklischen Abhängigkeiten)

Definition:

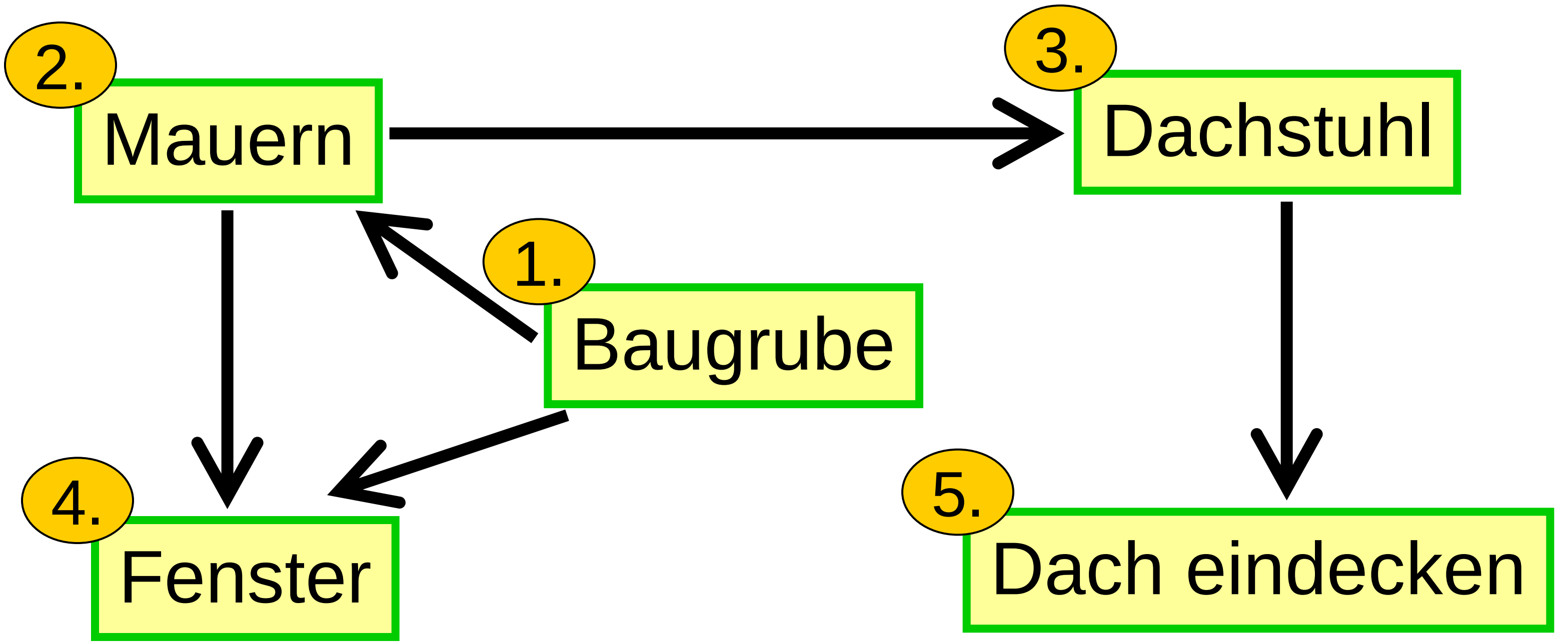
Ein *DAG (directed acyclic graph)* ist ein gerichteter Graph, der keinen (gerichteten) Kreis enthält.

Topologisches Sortieren

Topologisches Sortieren	
Gegeben:	DAG $G = (V,A)$
Gesucht:	eine Sortierung $v_1,\ldots,v_n$ der Knoten von G mit $i<j$ für alle $(v_i,v_j) \in A$



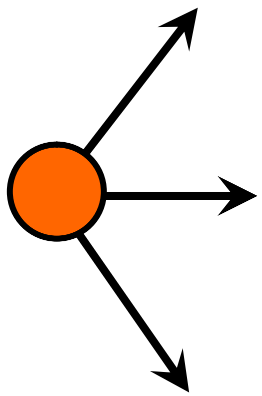
Warum ist dies nicht mit BFS lösbar?



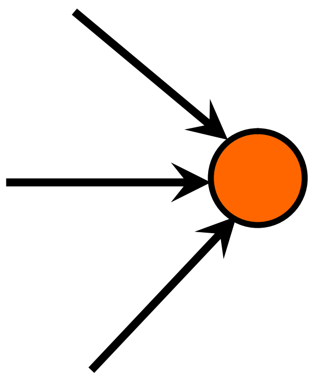
graphentheoretische Distanz von Baugrube zu Fenster = 1 (BFS), aber Fenster darf erst nach Mauer ausgeführt werden

Quellen & Senken

- Eine *Quelle* ist ein Knoten ohne eingehende Kanten
- Eine *Senke* ist ein Knoten ohne ausgehende Kanten



Quelle



Senke

Beobachtung:

Jeder (nicht-leere) DAG $G = (V,A)$ hat mindestens eine Quelle und eine Senke.

Beweis: Annahme: G hat *keine* Senke

- Verfolge von u_1 an immer ausgehende Kanten $p_i := u_1, u_2, \ldots, u_i$ solange bis $i > |V|$ oder keine ausgehende Kante mehr existiert
- Falls $i > |V|$ ➔ ein Knoten zweimal auf p_i
- ➔ Kreis! Widerspruch!
- Analog für Quelle

Algorithmus

Idee:
Wähle immer Quelle und entferne sie dann

while G ist nicht leer **do**

  Wähle eine Quelle s in G und gib sie aus

  $G := G - s$

end while

Verbesserungen

Wie finden wir effizient eine Quelle?

- Wird v gelöscht, dann können nur Zielknoten w von Kanten (v,w) neu zu Quellen werden.
- Genügt ausgehende Nachbarmenge $A^{-}(v)$ zu betrachten.

Müssen wir Knoten wirklich löschen?

- Nein!
  Verwalte Eingangsgrad in Knotenfeld *indeg*.

Algorithmus TopSort

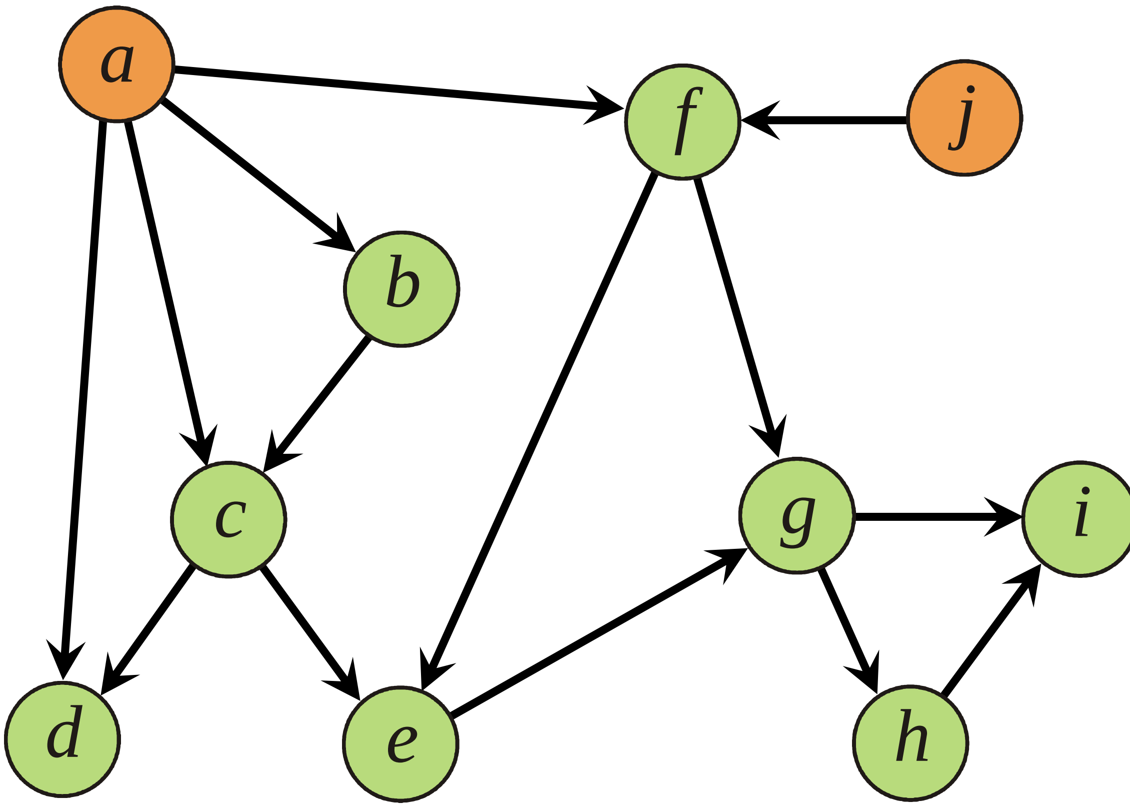
```
var Queue Q
var int indeg[V]

// Initialisierung
for all v ∈ V do {
    indeg[v] := d⁺(v)
    if indeg[v] = 0 then Q.PUT(v)
}
```

Algorithmus TopSort (2)

```
// Hauptschleife
while not Q.IsEmpty() do {
    v := Q.Get()
    Gib v aus
    for all (v,u) ∈ A⁻(v) do {
        indeg[u] := indeg[u] - 1
        if indeg[u] = 0 then Q.Put(u)
    }
}
```

Beispiel: TopSort



Ausgabe:

a
j
b
f
c
e
d
g
h
i

Q

a	j	b	f	c	e	d	g	h	i

Analyse der Laufzeit

- Initialisierung: $\Theta(|V|)$
  (da $d^+(v)$ in konstanter Zeit abrufbar)
- Jeder Knoten kommt genau einmal in Q
  ➔ **while**-Schleife wird $|V|$-mal durchlaufen
- Die **for all**-Schleife (Zeile 12) wird insgesamt für jede Kante einmal durchlaufen: $\Theta(|A|)$
- Gesamtaufwand: $\Theta(|V|+|A|)$

Analyse TopSort

Theorem:

Der Algorithmus TopSort berechnet eine topologische Sortierung der Knoten eines DAGs $G=(V,A)$ in Zeit $\Theta(|V|+|A|)$.