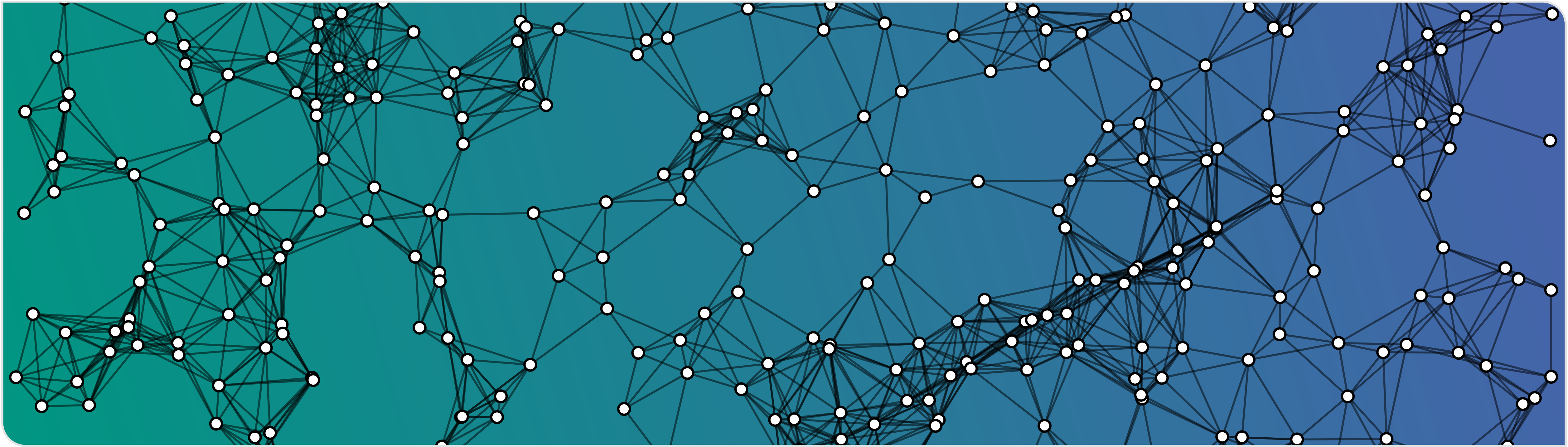
Karlsruher Institut für Technologie

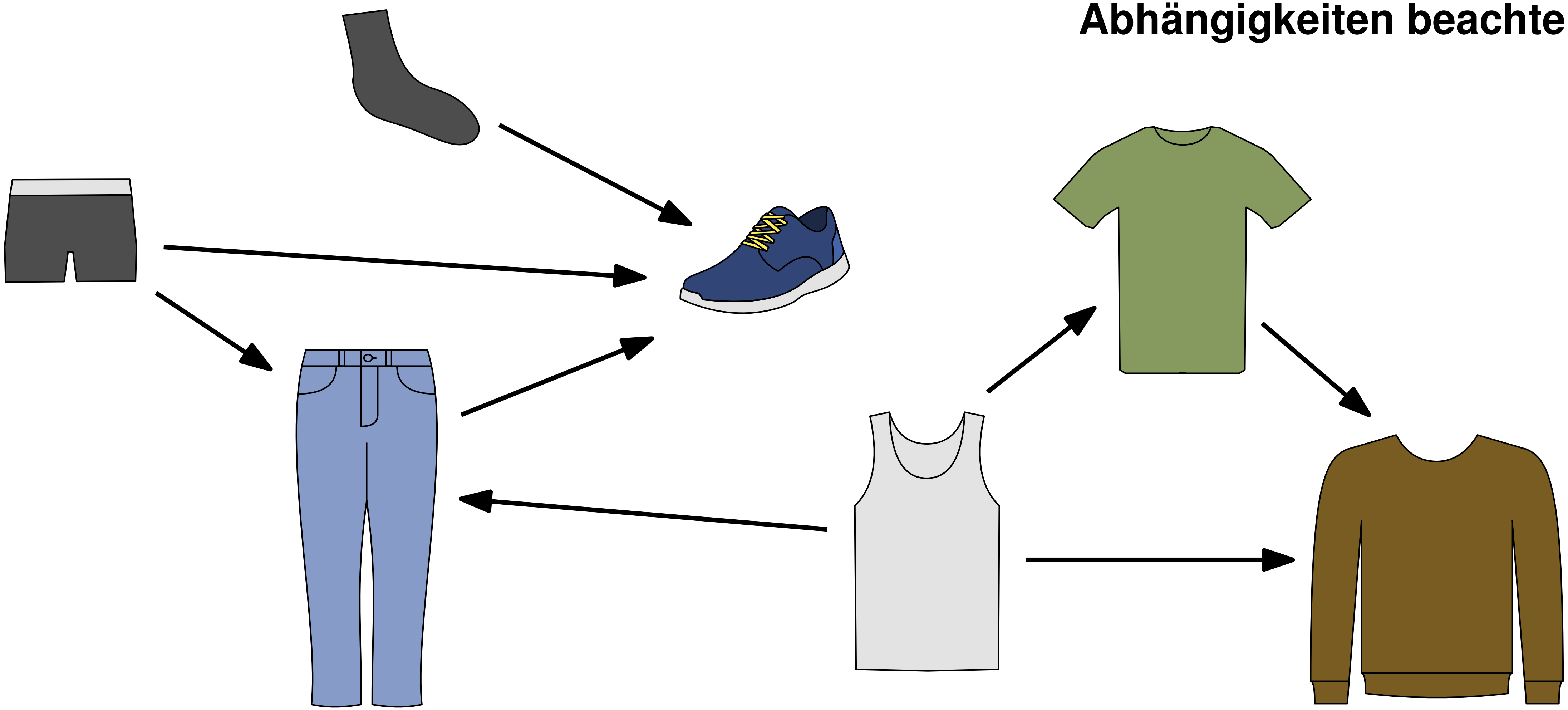
Algorithmen 1

Tiefensuche auf gerichteten Graphen
Topologische Sortierung



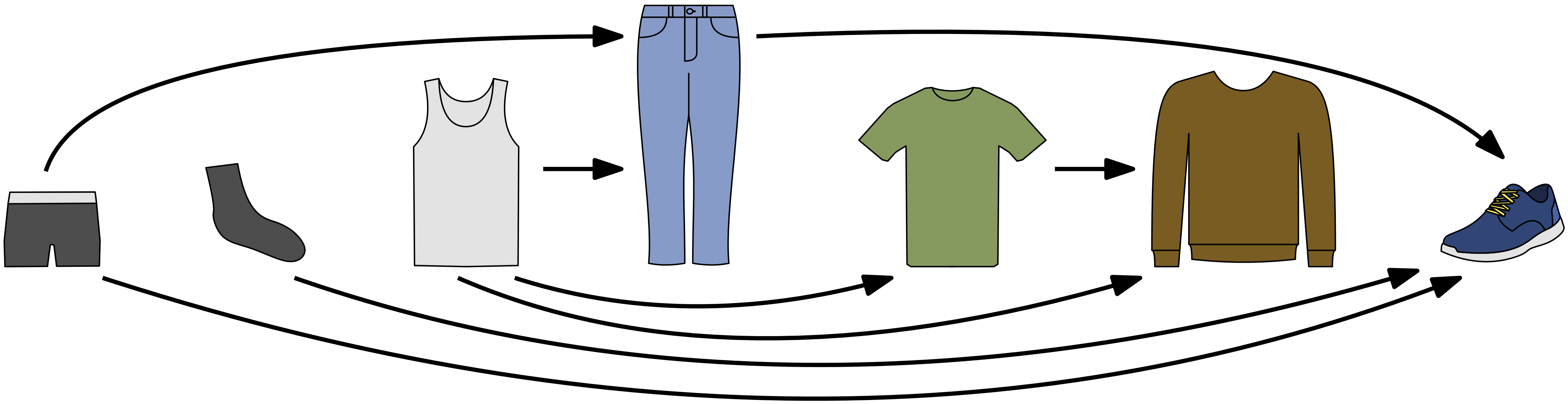
In welcher Reihenfolge sollte ich mich anziehen?

Abhängigkeiten beachten!



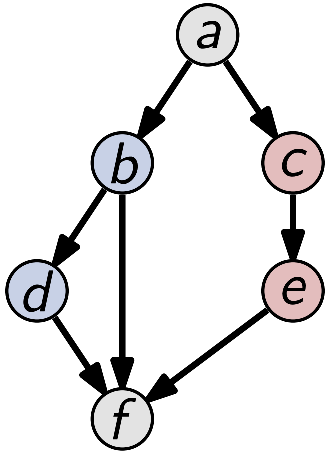
Topologische Sortierung

Definition
Sei $G = (V, E)$ ein gerichteter Graph. Eine **topologische Sortierung** ist eine total Ordnung der Knoten V, sodass jede Kante von kleinerem zu größerem Knoten zeigt.



Topologische Sortierung

Definition
Sei $G = (V, E)$ ein gerichteter Graph. Eine **topologische Sortierung** ist eine total Ordnung der Knoten V, sodass jede Kante von kleinerem zu größerem Knoten zeigt.



Ordnungen zählen

- a ist in jeder topologischen Sortierung der erste Knoten
- f ist in jeder topologischen Sortierung der letzte Knoten
- nach a kommt b oder c $\rightarrow$ die beiden Fälle sind symmetrisch
- drei topologische Sortierungen für jeden der Fälle

a b d c e f	a c e b d f
a b c d e f	a c b e d f
a b c e d f	a c b d e f

Anmerkung

- existiert genau dann wenn G azyklisch ist
- **DAG** – directed acyclic graph

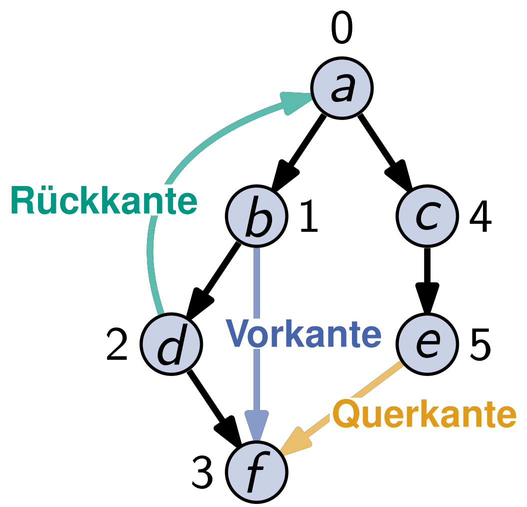
DFS auf gerichteten Graphen

Grundsätzliches Ziel auf ungerichteten Graphen

- starte bei einem Knoten s
- besuche alle von s aus erreichbaren Knoten
- also alle Knoten in der Zusammenhangskomponente von s

Grundsätzliches Ziel auf gerichteten Graphen

- starte bei einem Knoten s
- besuche alle von s aus erreichbaren Knoten
- benutze Kanten nur in die richtige Richtung



Beobachtungen

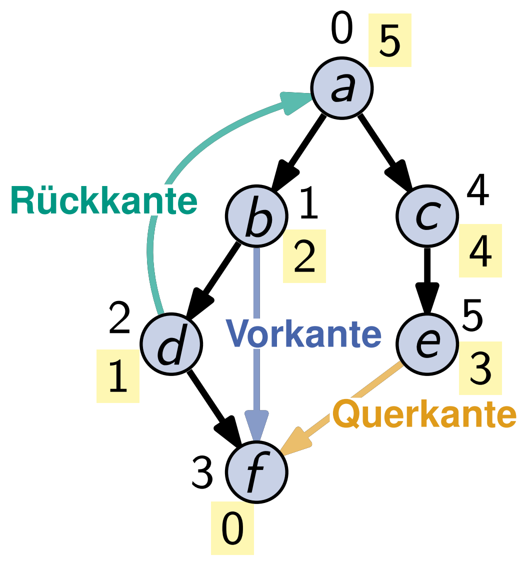
- man findet ggf. nicht alle Knoten in der Komponente (z.B.: $s = c$)
- **Rückkante** ist für manche nicht-Baumkanten eine unpassende Bezeichnung
  - **Vorkanten**: nicht-Baumkante zu einem Nachfolger im selben Teilbaum
  - **Querkanten**: Kante zu einem vorher betrachteten Teilbaum

Erkennung der verschiedenen Kantentypen

Typen von nicht-Baumkanten
- **Rückkante**: Kante zu einem Vorgänger im DFS-Baum
- **Vorkanten**: Kante zu einem Nachfolger im selben Teilbaum
- **Querkanten**: Kante zu einem vorher abgearbeiteten Teilbaum

Unterscheidung anhand der DFS-Nummer
- **Vorkanten** gehen von kleiner zu großer DFS-Nummer
- **Rückkanten** und **Querkanten** von groß zu klein

Unterschied zwischen Rück- und Querkanten
- **Rückkanten**: Start- ist vor Zielknoten fertig abgearbeitet
- **Querkante**: Start- ist nach Zielknoten fertig abgearbeitet
- speichere zusätzlich eine Fertigstellungs-Nummer: FIN-Nummer

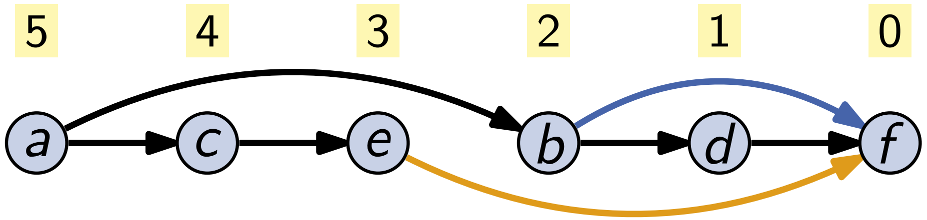


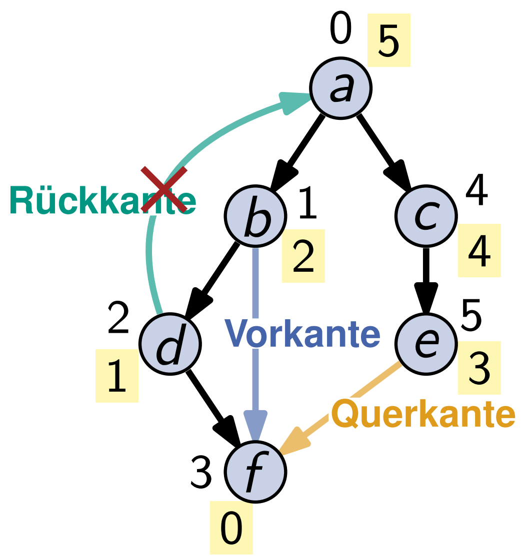
Wie hilft die DFS eine topologische Sortierung zu finden?

Karlsruher Institut für Technologie

Beobachtungen

- Rückkanten erzeugen einen Kreis $\rightarrow$ kann es im DAG nicht geben
- FIN-Nummer auf den übrigen Kanten:
  - geht immer von groß nach klein
  - ordne Knoten mit absteigender FIN-Nummer
    $\Rightarrow$ Kanten gehen alle von vorne nach hinten





Offene Frage

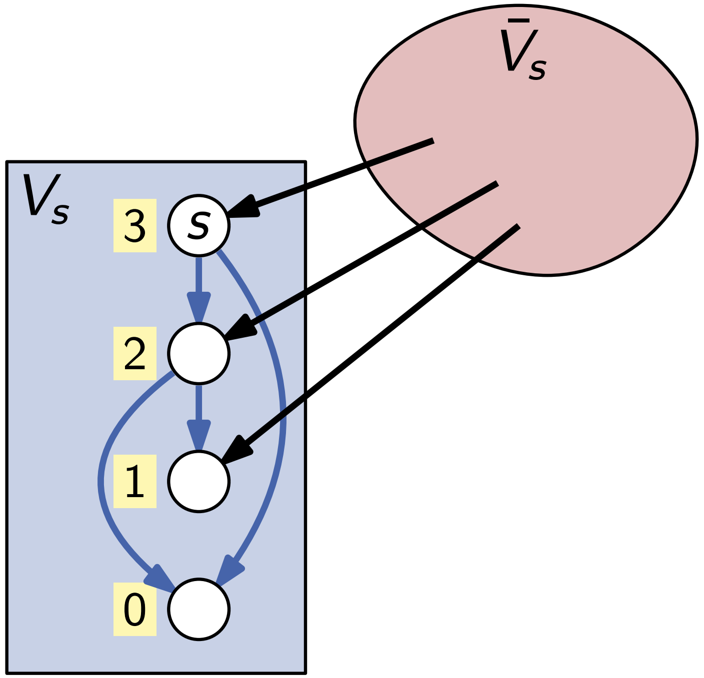
- ggf. sind nicht alle Knoten vom Start erreichbar
- Was tun wir dann?

	DFS-Nummer	FIN-Nummer
Vorkante	klein $\rightarrow$ groß	groß $\rightarrow$ klein
~~Rückkante~~	~~groß $\rightarrow$ klein~~	~~klein $\rightarrow$ groß~~
Querkante	groß $\rightarrow$ klein	groß $\rightarrow$ klein
Baumkante	klein $\rightarrow$ groß	groß $\rightarrow$ klein

Abarbeitung aller Knoten

Erste DFS

- starte DFS bei irgendeinem Knoten s
- sei V_s die von s aus erreichbare Knotenmenge
- FIN-Nummern liefern topologische Sortierung für V_s



Noch nicht erreichte Knoten

- seien $\bar{V}_s = V \setminus V_s$ die restlichen Knoten
- keine Kante von V_s nach $\bar{V}_s$ (DFS hätte sie sonst gefunden)
- es ist ok, wenn alle Knoten aus $\bar{V}_s$ vor V_s kommen

Weitere DFSs

- solange es noch unbesuchte Knoten gibt: DFS von unbesuchtem Knoten
- zähle FIN-Nummern weiter hoch
- FIN-Nummern liefern topologische Sortierung für alle Knoten

Algorithmus → Pseudocode

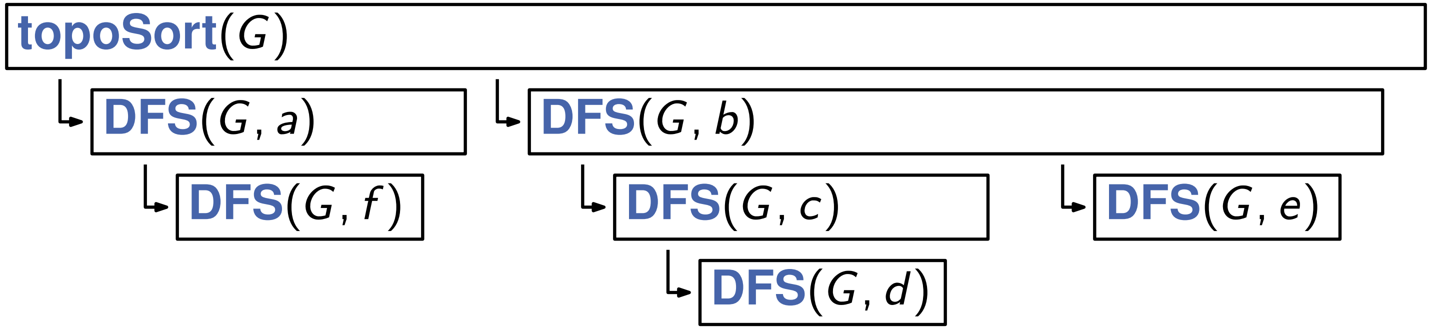
topoSort(*Graph* $G = (V, E)$)

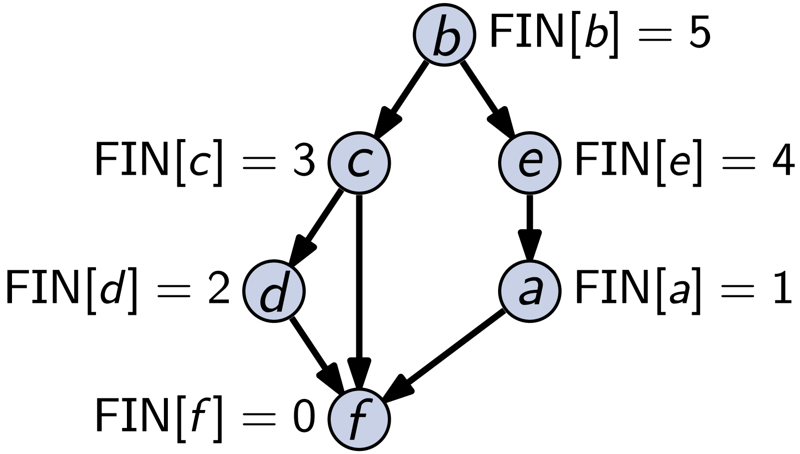
FIN := *Array* of size n initialized with ∞
curr := 0
for *Node* v in V **do**
  if v is uncolored **then**
    DFS(G, v)
return V sorted by decreasing FIN

DFS(*Graph* G, *Node* v)

color v
for *Node* u in $N(v)$ **do**
  if u is uncolored **then**
    DFS(G, u)
FIN$[v]$:= curr
curr := curr + 1

curr $= 6$





Laufzeit und Korrektheit

Theorem
Für einen DAG $G = (V, E)$ berechnet **topoSort** in $\Theta(n + m)$ eine topologische Sortierung.

Laufzeit

- grundsätzlich: **DFS** wird nur für ungefärbte Knoten aufgerufen
- Knoten wird gleich zu Beginn von **DFS** gefärbt
- daher: **DFS** wird für jeden Knoten einmal aufgerufen $\rightarrow \Theta(n)$ Aufrufe
- Kosten der **DFS** Aufrufe selbst:
  - **DFS**(G, v) iteriert über alle ausgehenden Kanten von v
  - das betrachtet insgesamt jede Kante einmal $\rightarrow \Theta(m)$
- zusätzlich: **topoSort** iteriert einmal über alle Knoten $\rightarrow \Theta(n)$

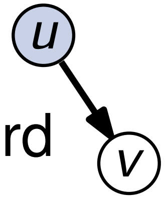
Laufzeit und Korrektheit

Theorem
Für einen DAG $G = (V, E)$ berechnet **topoSort** in $\Theta(n + m)$ eine topologische Sortierung.

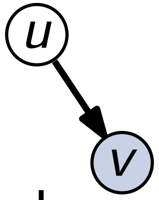
Korrektheit

Korrektheit folgt im Prinzip schon aus der Herleitung mit den Kantentypen von vorhin. Hier aber jetzt nochmal der Vollständigkeit halber ein formaler Beweis mit dem finalen Wissen, was der Algo tut.

- für eine Kante $(u, v) \in E$, zeige: $\mathrm{FIN}[u] > \mathrm{FIN}[v]$
- Fall 1: **DFS**(G, u) wird vor **DFS**(G, v) aufgerufen
  - **DFS**(G, v) wird aufgerufen, bevor **DFS**(G, u) fertig wird
  - **DFS**(G, v) wird fertig, bevor **DFS**(G, u) fertig wird
  - daher: $\mathrm{FIN}[u] > \mathrm{FIN}[v]$



- Fall 2: **DFS**(G, v) wird vor **DFS**(G, u) aufgerufen
  - G ist ein DAG $\Rightarrow$ u von v aus nicht erreichbar
  - **DFS**(G, v) wird fertig, bevor **DFS**(G, u) aufgerufen wird
  - daher: $\mathrm{FIN}[u] > \mathrm{FIN}[v]$

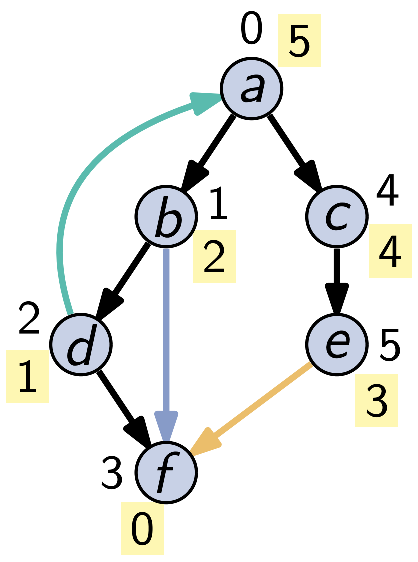



```
DFS(Graph G, Node v)
    color v
    for Node u in N(v) do
        if u is uncolored then
            DFS(G, u)
    FIN[v] := curr
    curr := curr + 1
```

Anmerkungen

Was passiert, wenn der Graph kein DAG ist?

- es werden trotzdem in $\Theta(n + m)$ die korrekten FIN-Nummern berechnet
- resultierende Ordnung: Rückkanten gehen in die falsche Richtung



Folgerung

- wir können **topoSort** also auch nutzen um zu testen, ob G ein DAG ist
- einfach **topoSort** laufen lassen und schauen, ob es Rückkanten gibt
- wenn zyklisch, dann findet man sogar einen Kreis: Rückkante + Pfad aus Baumkanten

	DFS-Nummer	FIN-Nummer
Vorkante	klein $\rightarrow$ groß	groß $\rightarrow$ klein
Rückkante	groß $\rightarrow$ klein	klein $\rightarrow$ groß
Querkante	groß $\rightarrow$ klein	groß $\rightarrow$ klein
Baumkante	klein $\rightarrow$ groß	groß $\rightarrow$ klein

Tiefensuche auf gerichteten Graphen

DFS – Tiefensuche

- grundlegender Algorithmus: Basis vieler anderer Algorithmen
- vier Kantentypen in gerichteten Graphen
- rekursive Implementierung
- Ausblick: Grundlage für starke Zusammenhangskomponenten

Ein bisschen Graphentheorie

- topologische Sortierung und DAGs
- strukturelle Einsicht $\rightarrow$ effizienter Algorithmus