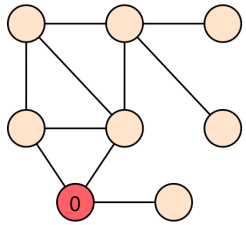
2.6 Graphen

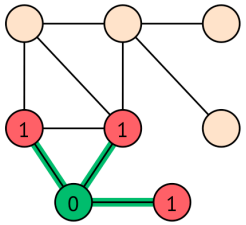
2.6.2 Ausspähen von Graphen

Breitensuche



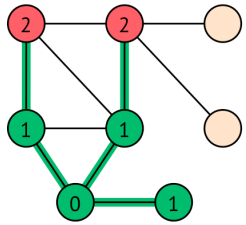
Quelle

Breitensuche



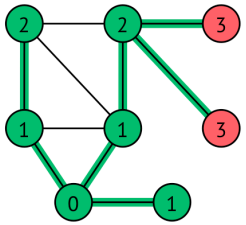
Quelle

Breitensuche



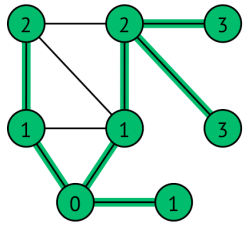
Quelle

Breitensuche



Quelle

Breitensuche



Quelle

Breitensuche

- Von einem gegebenen Startknoten s (Quelle, source) ausgehend, werden nacheinander alle von s erreichbaren Knoten besucht.
- Die Grenze zwischen besuchten und nicht besuchten Knoten wird gleichmäßig vorangetrieben (breadth-first)

Breitensuche

- Während der Breitensuche wird implizit oder explizit ein Breitensuchbaum aufgebaut.
- Die Tiefe eines Knotens u im Breitensuchbaum ist gleich der Länge des kürzesten Pfades von s zu u.
- Implementierung der Front durch eine Queue.

Analyse

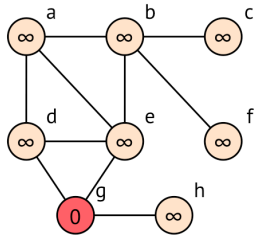
```
• BreadthFirstSearch( G )
    for all v ∈ V do
      depth[ v ] ← ∞
    for all v ∈ V do
      if depth[ v ] = ∞ then
        VisitBreadthFirst( G, v )
```


Der Aufruf im letzten if-Zweig bewirkt, dass alle Zusammenhangskomponenten des Graphen abgedeckt werden.

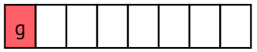
Analyse

- VISITBREADTHFIRST(G, s)
  depth[s] ← 0
  Q ← {}
  enqueue(Q, s)
  while Q not empty do
    u ← dequeue(Q)
    for all v ∈ Adj[u] do
      if depth[v] = ∞ then
        depth[v] ← depth[u] + 1
        enqueue(Q, v)

Breitensuche



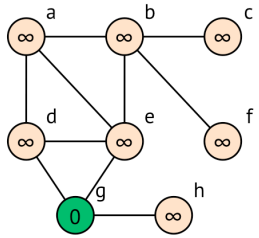
Q



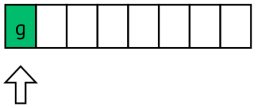
Quelle

Q: Warteschlange (FIFO)

Breitensuche

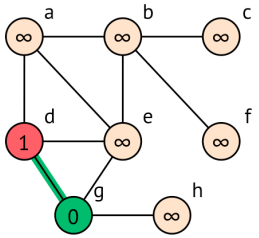


Q

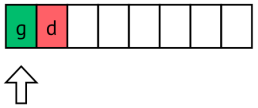


Quelle

Breitensuche

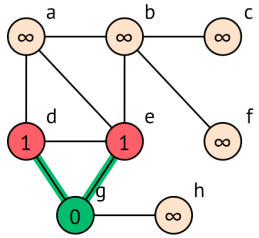


Q

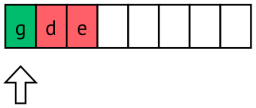


Quelle

Breitensuche

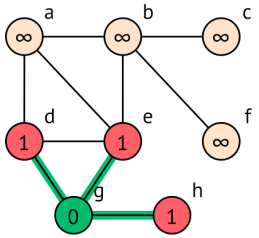


Q

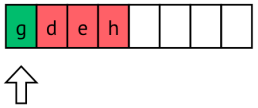


Quelle

Breitensuche

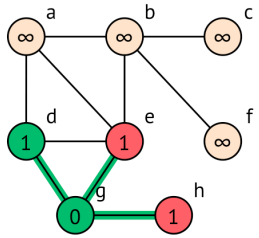


Q

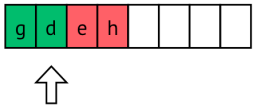


Quelle

Breitensuche

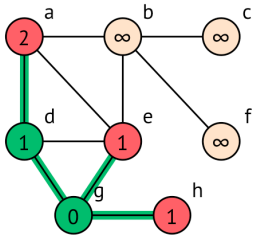


Q

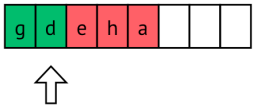


Quelle

Breitensuche

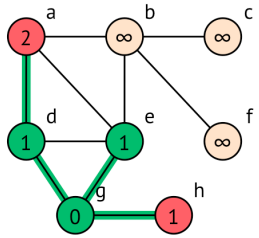


Q

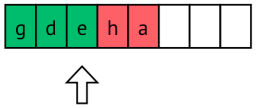


Quelle

Breitensuche

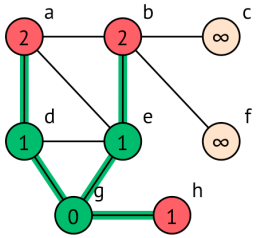


Q

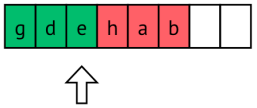


Quelle

Breitensuche

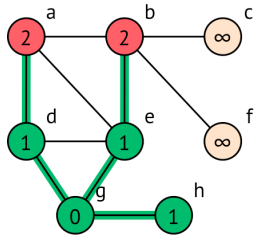


Q

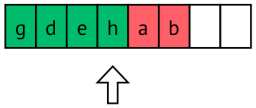


Quelle

Breitensuche

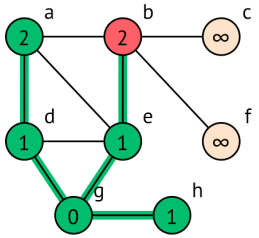


Q

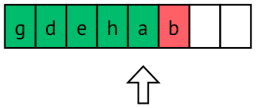


Quelle

Breitensuche

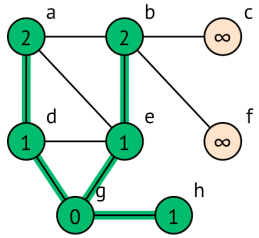


Q

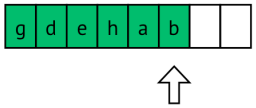


Quelle

Breitensuche

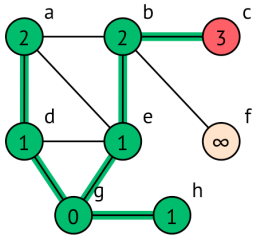


Q

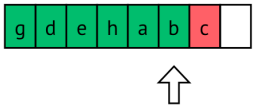


Quelle

Breitensuche

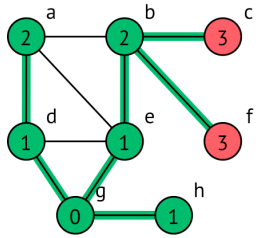


Q

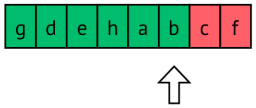


Quelle

Breitensuche

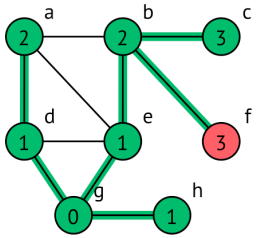


Q

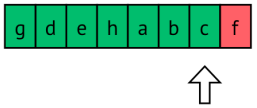


Quelle

Breitensuche

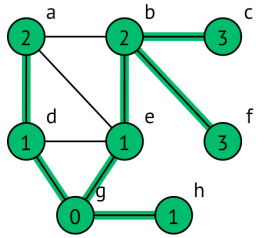


Q



Quelle

Breitensuche

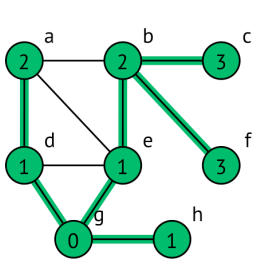


Q

g	d	e	h	a	b	c	f

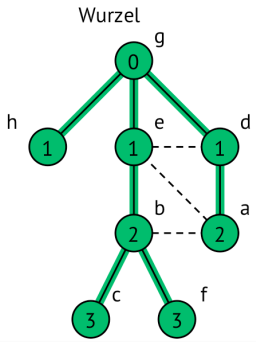
Quelle

Breitensuche



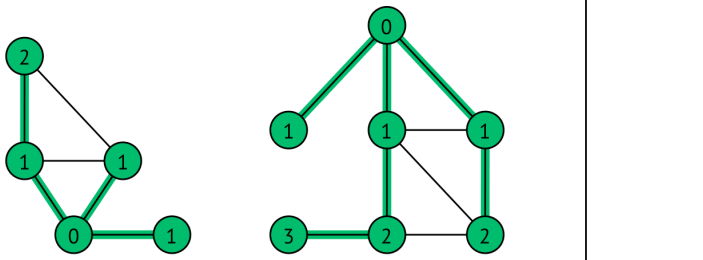
Quelle

Wurzel



Breitensuche extrahiert Baumstruktur aus zusammenhängendem Graphen und einen Wald aus einem Graphen, der aus mehreren Zusammenhangskomponenten besteht.

Breitensuchwald



Analyse

- BREADTHFIRSTSEARCH(G)
  - for all v ∈ V do
    - depth[v] ← ∞
  - for all v ∈ V do
    - if depth[v] = ∞ then
      - VISITBREADTHFIRST(G, v)
- Aufwand: O(V)

Breitensuche extrahiert Waldstruktur aus unzusammenhängendem Graphen

Analyse

- VISITBREADTHFIRST(G, s)
  depth[s] ← 0
  Q ← {}
  enqueue(Q, s)
  while Q not empty do
    u ← dequeue(Q)
    for all v ∈ Adj[u] do
      if depth[v] = ∞ then
        depth[v] ← depth[u] + 1
        enqueue(Q, v)

Wie oft werden diese innersten Anweisungen ausgeführt?

Analyse

- Jeder Knoten wird genau einmal in die Queue geschoben.
- Jeder Knoten wird genau einmal aus der Queue genommen.
- Adj[u] wird genau dann abgearbeitet, wenn u aus der Queue genommen wird.
- Der Aufwand zur Abarbeitung der Adjazenzlisten ist also $\sum_{u \in V} |Adj[u]| = O(E)$

Analyse

- Gesamtaufwand der Breitensuche
  $O(V+E)$

Eigenschaften

- Die Abstände zwischen allen Knotenpaaren eines Graphen lassen sich in Zeit $O(V\times(V+E))$ berechnen.
- Die Zahl der Zusammenhangs-komponenten eines ungerichteten Graphen läßt sich in $O(V+E)$ berechnen.
- Jeder ungerichtete Graph kann in $O(V+E)$ auf Zusammenhang getestet werden.

Mache für jeden Knoten Breitensuche, um kürzeste Abstände zwischen allen Knoten zu ermitteln
Zusammenhangskomponenten: Zähle Aufrufe der BreadthFirstSearch Prozedur
Zusammenhangstest: 1 Aufruf von BreadthFirstSearch

Breitensuche

- COMPONENTS(G)
  - for all v ∈ V do
    - comp[v] ← 0
  - components ← 0
  - for all v ∈ V do
    - if comp[v] = 0 then
      - components ← components + 1
      - VISITBREADTHFIRST'(G, v)
  - return components

Breitensuche

- VISITBREADTHFIRST'(G, s)
  - comp[s] ← components
  - Q ← {}
  - enqueue(Q, s)
  - while Q not empty do
    - u ← dequeue(Q)
    - for all v ∈ Adj[u] do
      - if comp[v] = 0 then
        - comp[v] ← components
        - enqueue(Q, v)

Eigenschaften

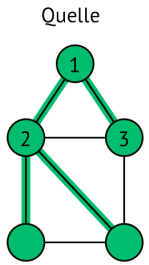
- Ein ungerichteter Graph lässt sich in $O(V)$ auf Kreisfreiheit testen.
- Erinnerung: Ein ungerichteter Graph $G=(V,E)$ mit k Zusammenhangskomponenten enthält einen Kreis, genau dann, wenn $E>V-k$
- Ist $E>V$ (nachzählen und abbrechen sobald Zähler größer V), so enthält G einen Kreis.
- Ist $E\leqslant V$ so zählt man die Komponenten wie oben beschrieben und zwar nur noch mit Aufwand $O(V+E)=O(V)$.

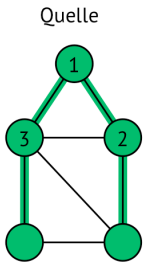
Ein Kreis ist eine Sequenz von Kanten, die einen Vertex mit sich selber verbindet. Dies kann man einem
Graphen bereits an der Anzahl der Kanten und Vertices "ansehen". Wenn ein zusammenhängender Graph
E = V – 1 Kanten hat, ist er kreisfrei. Sobald eine Kante mehr im Graphen ist, muss der Graph einen Kreis
enthalten.

Eigenschaften

- Breitensuche berechnet für jeden Knoten u die Länge des kürzesten Pfades von der Quelle s zu u.
- Der Breitensuchbaum ist nicht eindeutig, sondern hängt von der Reihenfolge der Knoten in den Adjazenzlisten ab.

Eigenschaften





Der durch Breitensuche entstehende Baum ist nicht eindeutig. Die Struktur ist davon abhängig,
in welcher Reihenfolge die Knoten besucht werden (gespeichert sind).

2.6.2 Ausspähen von Graphen

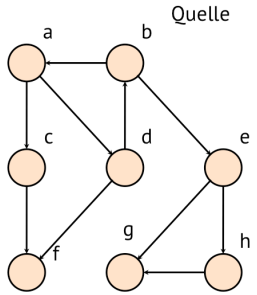
Tiefensuche

- Von einem gegebenen Startknoten s (Quelle, source) ausgehend, werden nacheinander alle von s erreichbaren Knoten besucht.
- Vom zuletzt besuchten Knoten werden zunächst die folgenden Knoten besucht (“depth-first”), Backtracking, falls alle Nachbarn besucht.

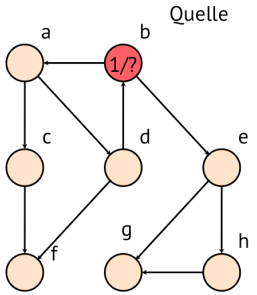
Tiefensuche

- In jedem Knoten werden die Eintritts- und Austrittszeitpunke gespeichert. Dies entspricht einer Präfix- bzw. Postfixsortierung auf dem entstehenden Tiefensuchwald.

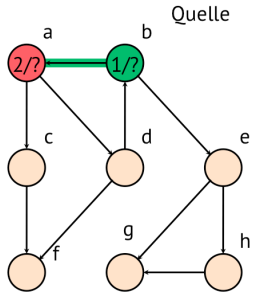
Beispiel



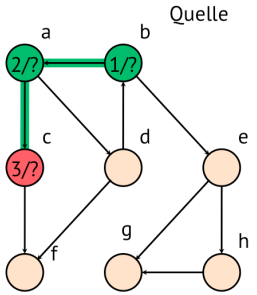
Beispiel



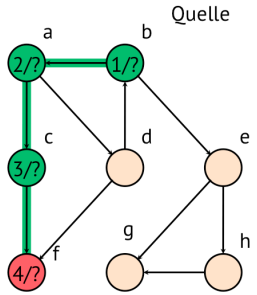
Beispiel



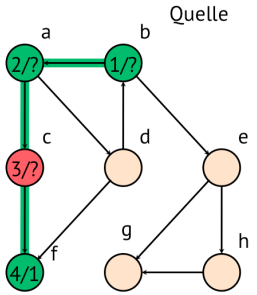
Beispiel



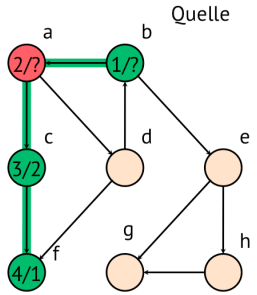
Beispiel



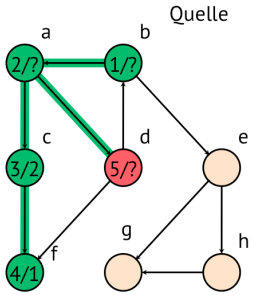
Beispiel



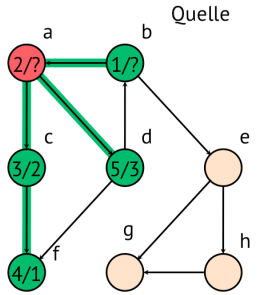
Beispiel



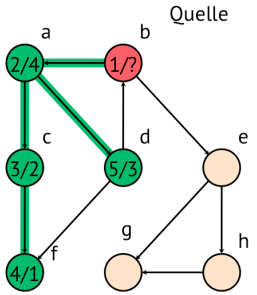
Beispiel



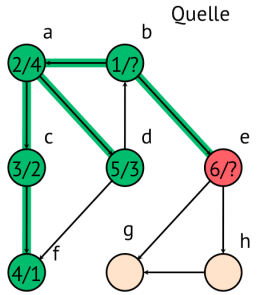
Beispiel



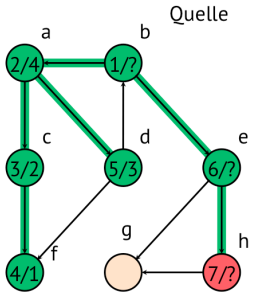
Beispiel



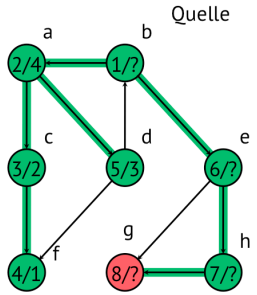
Beispiel



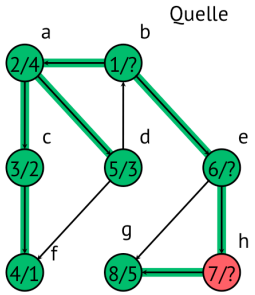
Beispiel



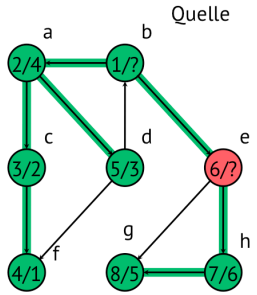
Beispiel



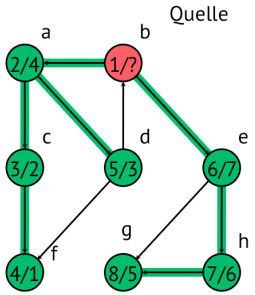
Beispiel



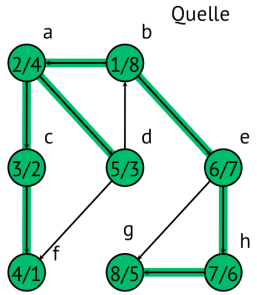
Beispiel



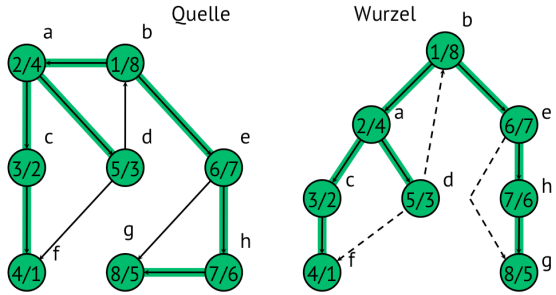
Beispiel



Beispiel

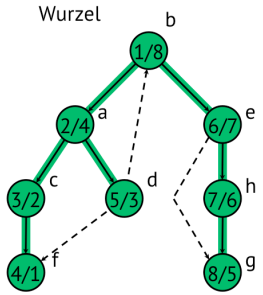


Tiefensuchbaum

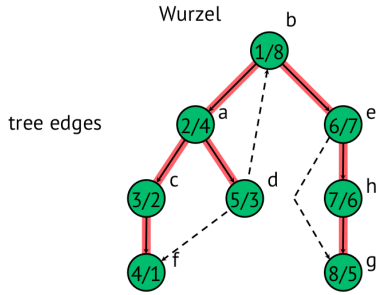


Auch ein Tiefensuchbaum ist nicht eindeutig aus demselben Grund, weshalb auch Breitensuchbäume nicht eindeutig sind.

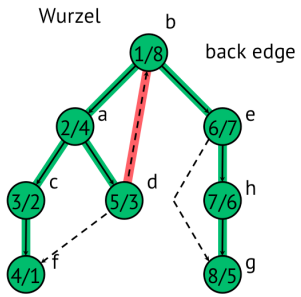
Tiefensuchbaum



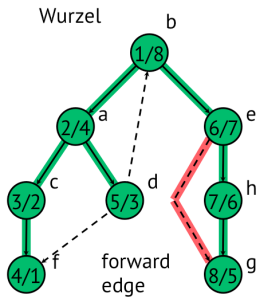
Tiefensuchbaum



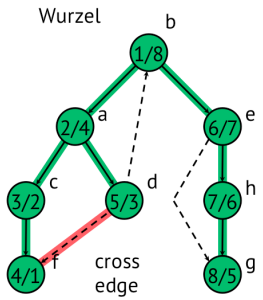
Tiefensuchbaum



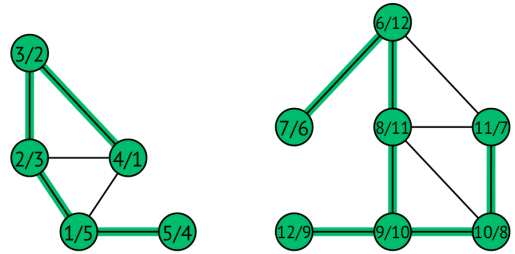
Tiefensuchbaum



Tiefensuchbaum



Tiefensuchwald



Tiefensuche

- DepthFirstSearch(G)
  for all v ∈ V do
    pre[v] ← 0
    post[v] ← 0
  prectr ← 0
  postctr ← 0
  for all v ∈ V do
    if pre[v] = 0 then
      DepthFirstVisit(v)

Tiefensuche

- DepthFirstVisit(u)
  prectr ← prectr + 1
  pre[u] ← prectr
  for all v∈Adj[u] do
    if pre[v] = 0 then
      DepthFirstVisit(v)
  postctr ← postctr + 1
  post[u] ← postctr

Die Abarbeitung erfolgt hier (im Gegensatz zur Breitensuche) in LIFO-Reihenfolge. Bei der Breitensuche erfolgt die Abarbeitung in FIFO-Reihenfolge.

Analyse

- DepthFirstSearch(G)
  for all v ∈ V do
    pre[v] ← 0
    post[v] ← 0
  prectr ← 0
  postctr ← 0
  for all v ∈ V do
    if pre[v] = 0 then
      DepthFirstVisit(v)
- Aufwand: O(V)

Analyse

- DepthFirstVisit(u)
  prectr ← prectr + 1
  pre[u] ← prectr
  for all v∈Adj[u] do
    if pre[v] = 0 then
      DepthFirstVisit(v)
  postctr ← postctr + 1
  post[u] ← postctr
- Aufwand: ...

Initialisierung -> Zähler auf null setzen für alle Knoten

Analyse

- Jeder Knoten wird genau einmal besucht.
- Adj[u] wird genau dann abgearbeitet, wenn u besucht wird.
- Der Aufwand zur Abarbeitung aller Adjazenzlisten ist also $\sum_{u \in V} |Adj[u]| = O(E)$

Analyse

- Gesamtaufwand der Tiefensuche
  O(V+E)

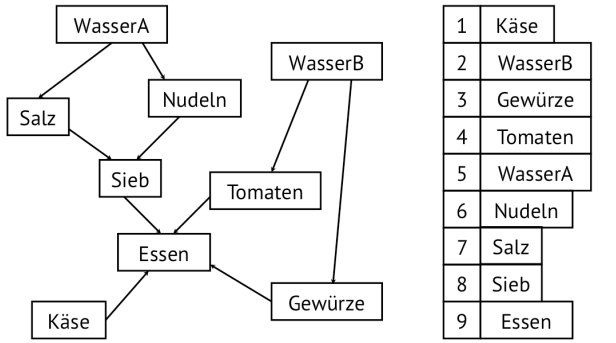
2.6.2 Ausspähen von Graphen

Topologisches Sortieren

- Eine vollständige Ordnung $(V,<)$ heißt eine topologische Sortierung des gerichteten, zyklenfreien Graphen $G=(V,E)$, falls gilt $(u,v) \in E \rightarrow u < v$

Man sucht eine Totalordnung (Sortierung aller Elemente) der Knoten in einem Graphen, die kompatibel mit dem Graph ist.
Dies geht natürlich nur dann, wenn der Graph zyklenfrei ist.

Topologisches Sortieren

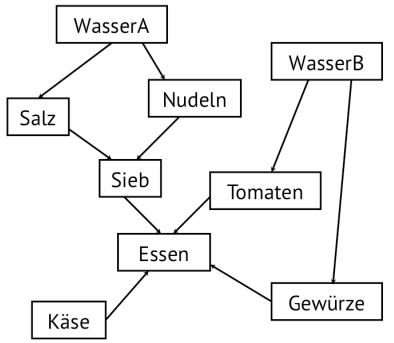


Wie kann man aus einem solchen Graphen die rechts dargestellte (sortierte) Sequenz extrahieren?

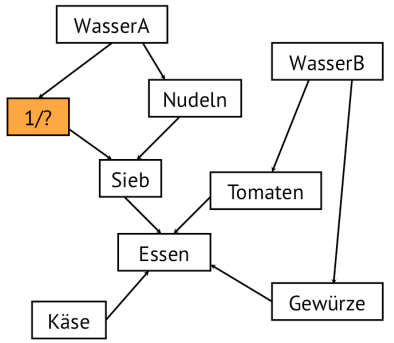
Topologisches Sortieren

- Es sei p die Postfixnummerierung des Tiefensuchwalds von G. Dann ist die Ordnung der Knoten v nach absteigendem p(v) eine topologische Sortierung von G.

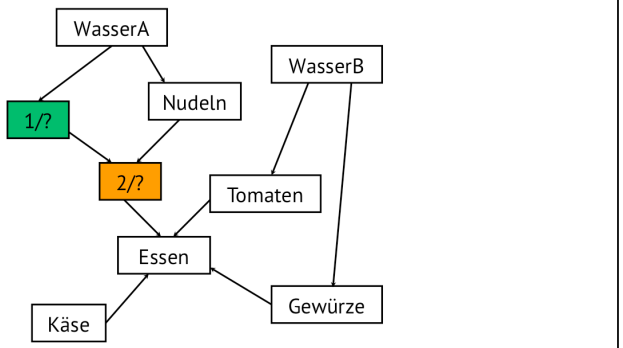
Topologisches Sortieren



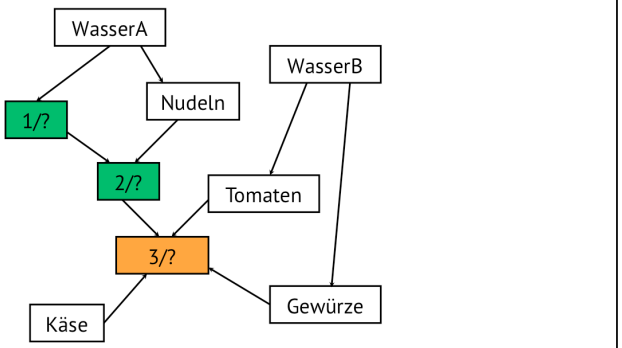
Topologisches Sortieren



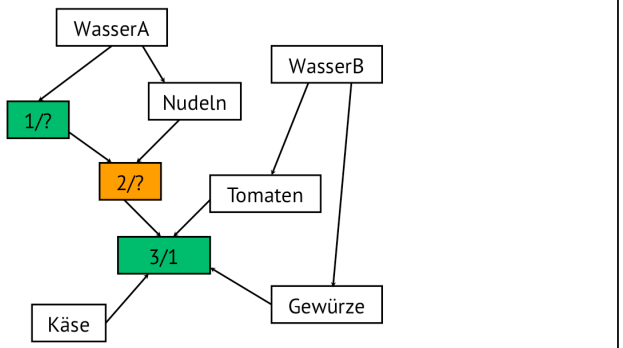
Topologisches Sortieren



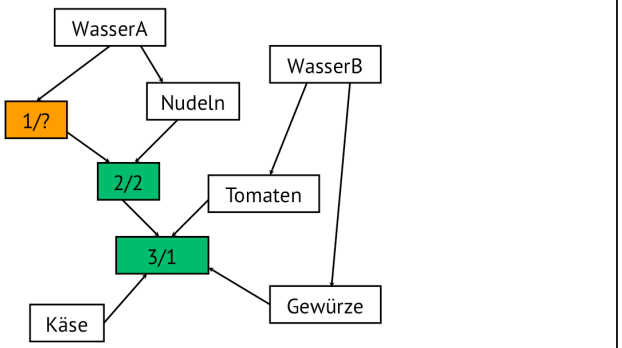
Topologisches Sortieren



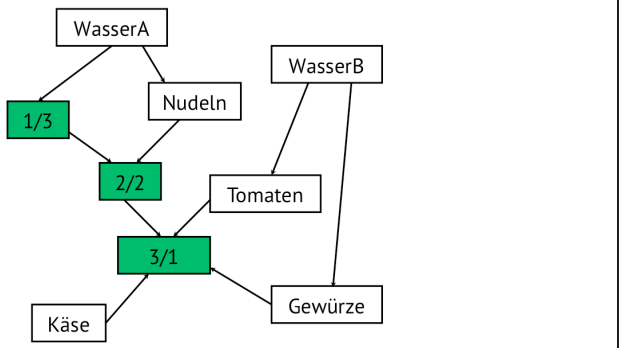
Topologisches Sortieren



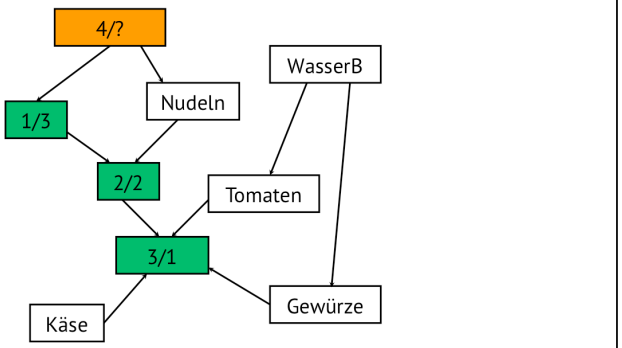
Topologisches Sortieren



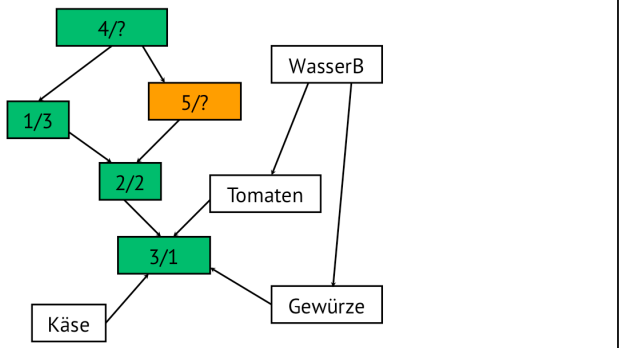
Topologisches Sortieren



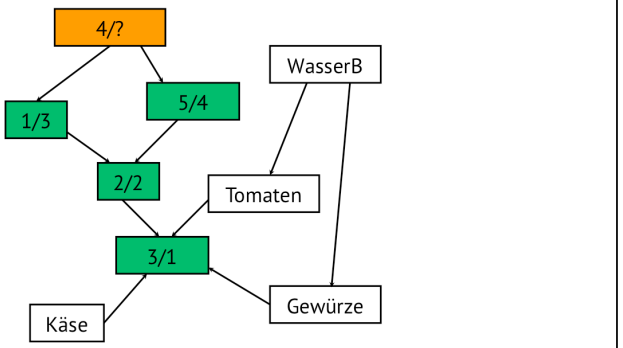
Topologisches Sortieren



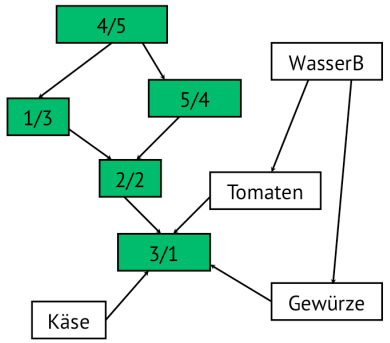
Topologisches Sortieren



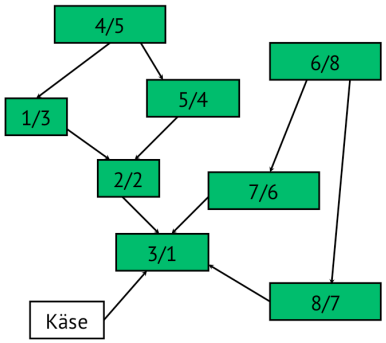
Topologisches Sortieren

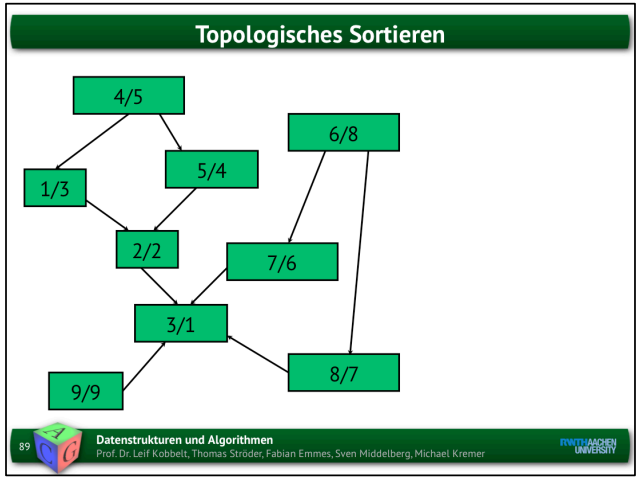


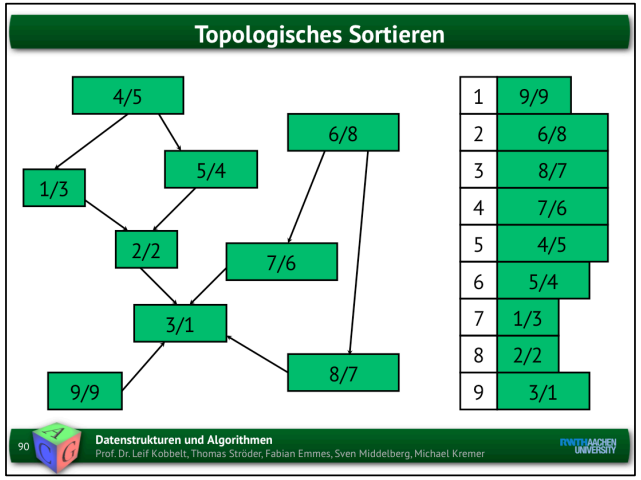
Topologisches Sortieren



Topologisches Sortieren

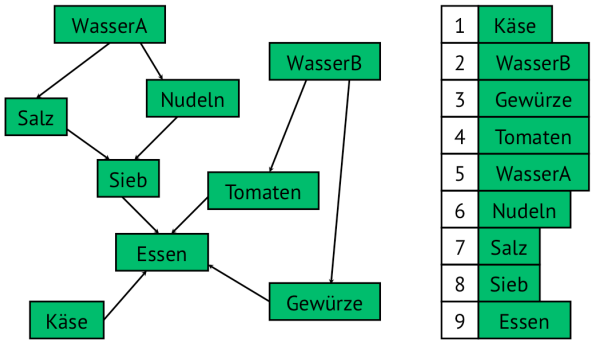






Aufzählen der Knoten in umgekehrter Reihenfolge ausgehend nach zweitem Schlüssel (Return).

Topologisches Sortieren



1	Käse
2	WasserB
3	Gewürze
4	Tomaten
5	WasserA
6	Nudeln
7	Salz
8	Sieb
9	Essen