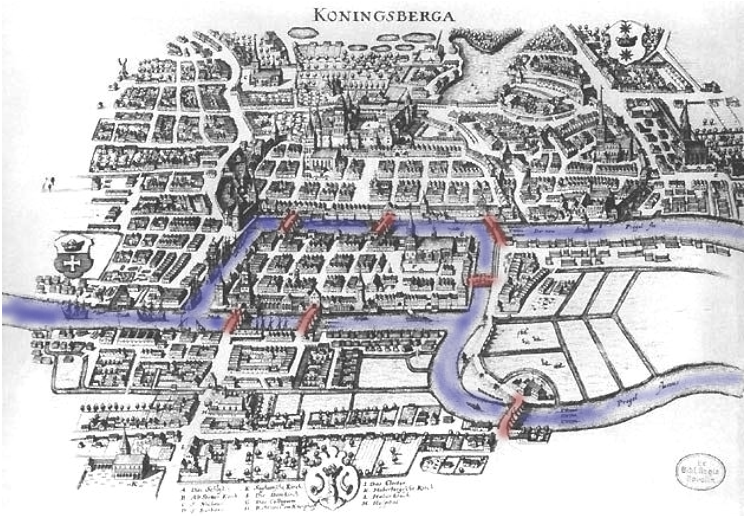
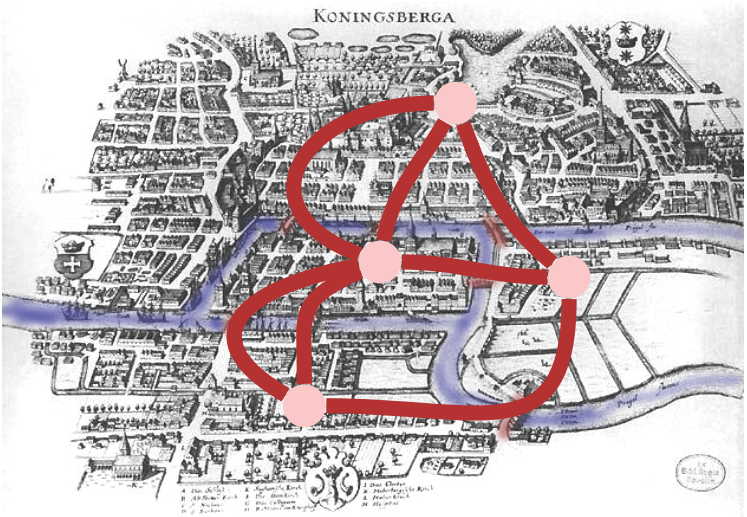
24. Graphen

Notation, Repräsentation, Traversieren (DFS, BFS), Topologisches Sortieren , Reflexive transitive Hülle, Zusammenhangskomponenten
[Ottman/Widmayer, Kap. 9.1 - 9.4,Cormen et al, Kap. 22]

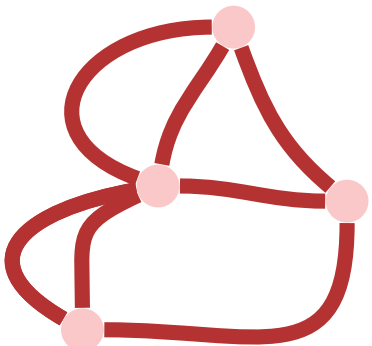
Königsberg 1736



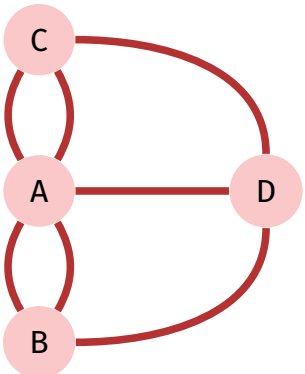
Königsberg 1736



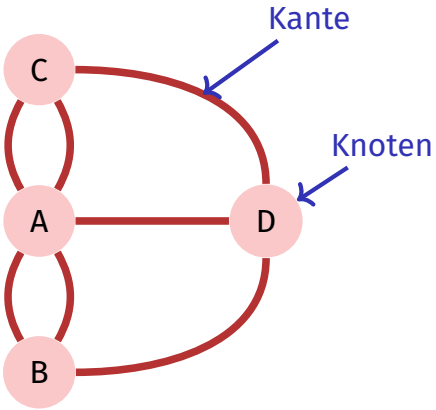
Königsberg 1736



[Multi]Graph

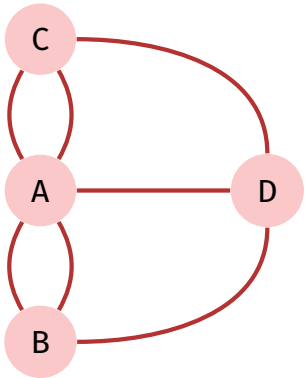


[Multi]Graph



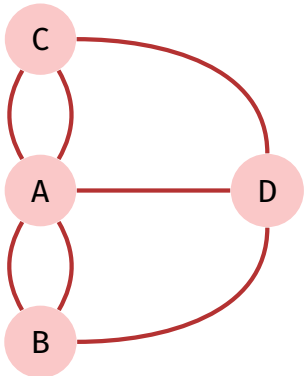
Zyklen

- Gibt es einen Rundweg durch die Stadt (den Graphen), welcher jede Brücke (jede Kante) genau einmal benutzt?



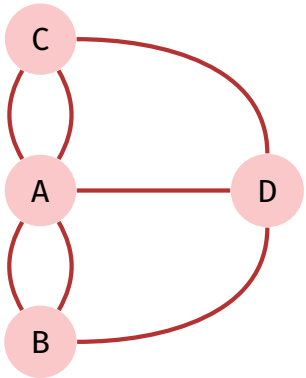
Zyklen

- Gibt es einen Rundweg durch die Stadt (den Graphen), welcher jede Brücke (jede Kante) genau einmal benutzt?
- Euler (1736): nein.



Zyklen

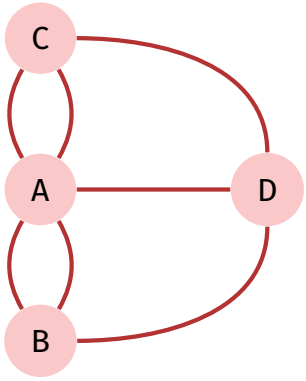
- Gibt es einen Rundweg durch die Stadt (den Graphen), welcher jede Brücke (jede Kante) genau einmal benutzt?
- Euler (1736): nein.
- Solcher Rundweg (*Zyklus*) heisst Eulerscher Kreis.



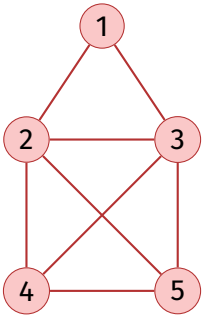
Zyklen

- Gibt es einen Rundweg durch die Stadt (den Graphen), welcher jede Brücke (jede Kante) genau einmal benutzt?
- Euler (1736): nein.
- Solcher Rundweg (*Zyklus*) heisst Eulerscher Kreis.
- Eulerzyklus $\Leftrightarrow$ jeder Knoten hat gerade Anzahl Kanten (jeder Knoten hat einen *geraden Grad*).

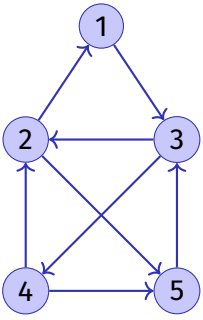
  "$\Rightarrow$" ist sofort klar, "$\Leftarrow$" ist etwas schwieriger, aber auch elementar.



Notation



ungerichtet



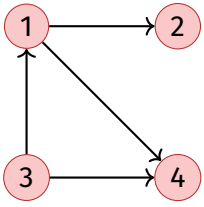
gerichtet

$$V = \{1, 2, 3, 4, 5\}$$
$$E = \{\{1, 2\}, \{1, 3\}, \{2, 3\}, \{2, 4\}, \{2, 5\}, \{3, 4\}, \{3, 5\}, \{4, 5\}\}$$

$$V = \{1, 2, 3, 4, 5\}$$
$$E = \{(1, 3), (2, 1), (2, 5), (3, 2), (3, 4), (4, 2), (4, 5), (5, 3)\}$$

Notation

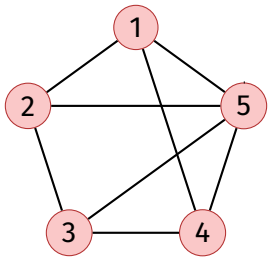
Ein gerichteter Graph besteht aus einer Menge $V = \{v_1, \ldots, v_n\}$ von Knoten (*Vertices*) und einer Menge $E \subseteq V \times V$ von Kanten (*Edges*). Gleiche Kanten dürfen nicht mehrfach enthalten sein.



Schleife

Notation

Ein ungerichteter Graph besteht aus einer Menge $V = \{v_1, \ldots, v_n\}$ von Knoten und einer Menge $E \subseteq \{\{u, v\} | u, v \in V\}$ von Kanten. Kanten dürfen nicht mehrfach enthalten sein.[39]

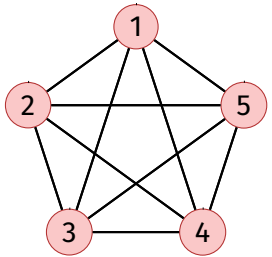


ungerichter Graph

[39] Im Gegensatz zum Eingangsbeispiel – dann Multigraph genannt.

Notation

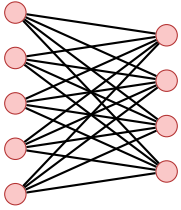
Ein ungerichteter Graph $G = (V, E)$ ohne Schleifen in dem jeder Knoten mit jedem anderen Knoten durch eine Kante verbunden ist, heisst vollständig.



ein vollständiger ungerichter Graph

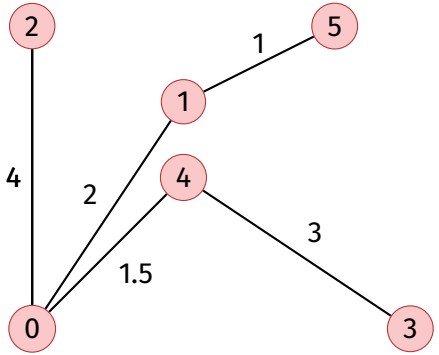
Notation

Ein Graph, bei dem V so in disjunkte U und W aufgeteilt werden kann, dass alle $e \in E$ einen Knoten in U und einen in W haben heisst bipartit.



Notation

Ein gewichteter Graph $G = (V, E, c)$ ist ein Graph $G = (V, E)$ mit einer Kantengewichtsfunktion $c : E \to \mathbb{R}$. $c(e)$ heisst Gewicht der Kante e.



Notation

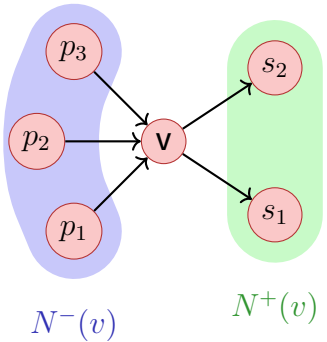
Für gerichtete Graphen $G = (V, E)$

- $w \in V$ heisst adjazent zu $v \in V$, falls $(v, w) \in E$

Notation

Für gerichtete Graphen $G = (V, E)$

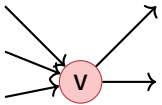
- $w \in V$ heisst adjazent zu $v \in V$, falls $(v, w) \in E$
- Vorgängermenge von $v \in V$: $N^-(v) := \{u \in V | (u, v) \in E\}$.
  Nachfolgermenge: $N^+(v) := \{u \in V | (v, u) \in E\}$



Notation

Für gerichtete Graphen $G = (V, E)$

- Eingangsgrad: $\deg^-(v) = |N^-(v)|$,
  Ausgangsgrad: $\deg^+(v) = |N^+(v)|$



$\deg^-(v) = 3$, $\deg^+(v) = 2$

$\deg^-(w) = 1$, $\deg^+(w) = 1$

Notation

Für ungerichtete Graphen $G = (V, E)$:

- $w \in V$ heisst adjazent zu $v \in V$, falls $\{v, w\} \in E$

Notation

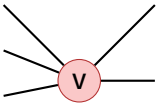
Für ungerichtete Graphen $G = (V, E)$:

- $w \in V$ heisst adjazent zu $v \in V$, falls $\{v, w\} \in E$
- Nachbarschaft von $v \in V$: $N(v) = \{w \in V | \{v, w\} \in E\}$

Notation

Für ungerichtete Graphen $G = (V, E)$:

- $w \in V$ heisst adjazent zu $v \in V$, falls $\{v, w\} \in E$
- Nachbarschaft von $v \in V$: $N(v) = \{w \in V | \{v, w\} \in E\}$
- Grad von v: $\deg(v) = |N(v)|$ mit Spezialfall Schleifen: erhöhen Grad um 2.



$\deg(v) = 5$

$\deg(w) = 2$

Beziehung zwischen Knotengraden und Kantenzahl

In jedem Graphen $G = (V, E)$ gilt

1. $\sum_{v \in V} \deg^-(v) = \sum_{v \in V} \deg^+(v) = |E|$, falls G gerichtet
2. $\sum_{v \in V} \deg(v) = 2|E|$, falls G ungerichtet.

Wege

- Weg: Sequenz von Knoten $\langle v_1, \ldots, v_{k+1} \rangle$ so dass für jedes $i \in \{1 \ldots k\}$ eine Kante von v_i nach v_{i+1} existiert.

Wege

- Weg: Sequenz von Knoten $\langle v_1, \ldots, v_{k+1} \rangle$ so dass für jedes $i \in \{1 \ldots k\}$ eine Kante von v_i nach v_{i+1} existiert.
- Länge des Weges: Anzahl enthaltene Kanten k.

Wege

- Weg: Sequenz von Knoten $\langle v_1, \ldots, v_{k+1} \rangle$ so dass für jedes $i \in \{1 \ldots k\}$ eine Kante von v_i nach v_{i+1} existiert.
- Länge des Weges: Anzahl enthaltene Kanten k.
- Gewicht des Weges (in gewichteten Graphen): $\sum_{i=1}^{k} c((v_i, v_{i+1}))$ (bzw. $\sum_{i=1}^{k} c(\{v_i, v_{i+1}\})$)

Wege

- Weg: Sequenz von Knoten $\langle v_1, \ldots, v_{k+1} \rangle$ so dass für jedes $i \in \{1 \ldots k\}$ eine Kante von v_i nach v_{i+1} existiert.
- Länge des Weges: Anzahl enthaltene Kanten k.
- Gewicht des Weges (in gewichteten Graphen): $\sum_{i=1}^{k} c((v_i, v_{i+1}))$ (bzw. $\sum_{i=1}^{k} c(\{v_i, v_{i+1}\})$)
- Pfad (auch: einfacher Pfad): Weg der keinen Knoten mehrfach verwendet.

Zusammenhang

- Ungerichteter Graph heisst zusammenhängend, wenn für jedes Paar $v, w \in V$ ein verbindender Weg existiert.
- Gerichteter Graph heisst stark zusammenhängend, wenn für jedes Paar $v, w \in V$ ein verbindender Weg existiert.
- Gerichteter Graph heisst schwach zusammenhängend, wenn der entsprechende ungerichtete Graph zusammenhängend ist.

Einfache Beobachtungen

- Allgemein: $0 \leq |E| \in \mathcal{O}(|V|^2)$
- Zusammenhängender Graph: $|E| \in \Omega(|V|)$
- Vollständiger Graph: $|E| = \frac{|V| \cdot (|V|-1)}{2}$ (ungerichtet)
- Maximal $|E| = |V|^2$ (gerichtet),$|E| = \frac{|V| \cdot (|V|+1)}{2}$ (ungerichtet)

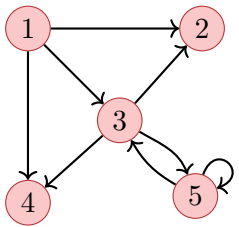
Zyklen

- Zyklus: Weg $\langle v_1, \ldots, v_{k+1} \rangle$ mit $v_1 = v_{k+1}$
- Kreis: Zyklus mit paarweise verschiedenen $v_1, \ldots, v_k$, welcher keine Kante mehrfach verwendet.
- Kreisfrei (azyklisch): Graph ohne jegliche Kreise.

Eine Folgerung: Ungerichtete Graphen können keinen Kreis der Länge 2 enthalten (Schleifen haben Länge 1).

Repräsentation mit Matrix

Graph $G = (V, E)$ mit Knotenmenge $v_1, \ldots, v_n$ gespeichert als Adjazenzmatrix $A_G = (a_{ij})_{1 \leq i,j \leq n}$ mit Einträgen aus $\{0, 1\}$. $a_{ij} = 1$ genau dann wenn Kante von v_i nach v_j.

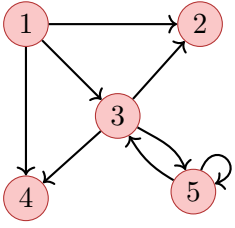


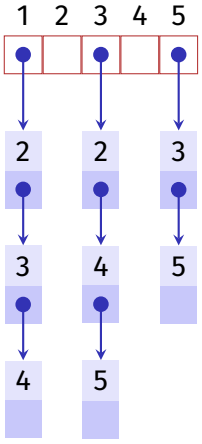
$$\begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix}$$

Speicherbedarf $\Theta(|V|^2)$. A_G ist symmetrisch, wenn G ungerichtet.

Repräsentation mit Liste

Viele Graphen $G = (V, E)$ mit Knotenmenge $v_1, \ldots, v_n$ haben deutlich weniger als n^2 Kanten. Repräsentation mit Adjazenzliste: Array $A[1], \ldots, A[n]$, A_i enthält verkettete Liste aller Knoten in $N^+(v_i)$.





Speicherbedarf $\Theta(|V| + |E|)$.

Laufzeiten einfacher Operationen

Operation	Matrix	Liste
Nachbarn/Nachfolger von $v \in V$ finden		
$v \in V$ ohne Nachbar/Nachfolger finden		
$(v, u) \in E$?		
Kante einfügen		
Kante (v, u) löschen		

Laufzeiten einfacher Operationen

Operation	Matrix	Liste
Nachbarn/Nachfolger von $v \in V$ finden	$\Theta(n)$	
$v \in V$ ohne Nachbar/Nachfolger finden		
$(v, u) \in E$?		
Kante einfügen		
Kante (v, u) löschen		

Laufzeiten einfacher Operationen

Operation	Matrix	Liste
Nachbarn/Nachfolger von $v \in V$ finden	$\Theta(n)$	$\Theta(\deg^+ v)$
$v \in V$ ohne Nachbar/Nachfolger finden		
$(v,u) \in E$?		
Kante einfügen		
Kante (v,u) löschen		

Laufzeiten einfacher Operationen

Operation	Matrix	Liste
Nachbarn/Nachfolger von $v \in V$ finden	$\Theta(n)$	$\Theta(\deg^+ v)$
$v \in V$ ohne Nachbar/Nachfolger finden	$\Theta(n^2)$	
$(v,u) \in E$?		
Kante einfügen		
Kante (v,u) löschen		

Laufzeiten einfacher Operationen

Operation	Matrix	Liste
Nachbarn/Nachfolger von $v \in V$ finden	$\Theta(n)$	$\Theta(\deg^+ v)$
$v \in V$ ohne Nachbar/Nachfolger finden	$\Theta(n^2)$	$\Theta(n)$
$(v,u) \in E$?		
Kante einfügen		
Kante (v,u) löschen		

Laufzeiten einfacher Operationen

Operation	Matrix	Liste
Nachbarn/Nachfolger von $v \in V$ finden	$\Theta(n)$	$\Theta(\deg^+ v)$
$v \in V$ ohne Nachbar/Nachfolger finden	$\Theta(n^2)$	$\Theta(n)$
$(v,u) \in E$?	$\Theta(1)$	
Kante einfügen		
Kante (v,u) löschen		

Laufzeiten einfacher Operationen

Operation	Matrix	Liste
Nachbarn/Nachfolger von $v \in V$ finden	$\Theta(n)$	$\Theta(\deg^+ v)$
$v \in V$ ohne Nachbar/Nachfolger finden	$\Theta(n^2)$	$\Theta(n)$
$(v,u) \in E$?	$\Theta(1)$	$\Theta(\deg^+ v)$
Kante einfügen		
Kante (v,u) löschen		

Laufzeiten einfacher Operationen

Operation	Matrix	Liste
Nachbarn/Nachfolger von $v \in V$ finden	$\Theta(n)$	$\Theta(\deg^+ v)$
$v \in V$ ohne Nachbar/Nachfolger finden	$\Theta(n^2)$	$\Theta(n)$
$(v,u) \in E$?	$\Theta(1)$	$\Theta(\deg^+ v)$
Kante einfügen	$\Theta(1)$	
Kante (v,u) löschen		

Laufzeiten einfacher Operationen

Operation	Matrix	Liste
Nachbarn/Nachfolger von $v \in V$ finden	$\Theta(n)$	$\Theta(\deg^+ v)$
$v \in V$ ohne Nachbar/Nachfolger finden	$\Theta(n^2)$	$\Theta(n)$
$(v,u) \in E$?	$\Theta(1)$	$\Theta(\deg^+ v)$
Kante einfügen	$\Theta(1)$	$\Theta(1)$
Kante (v,u) löschen		

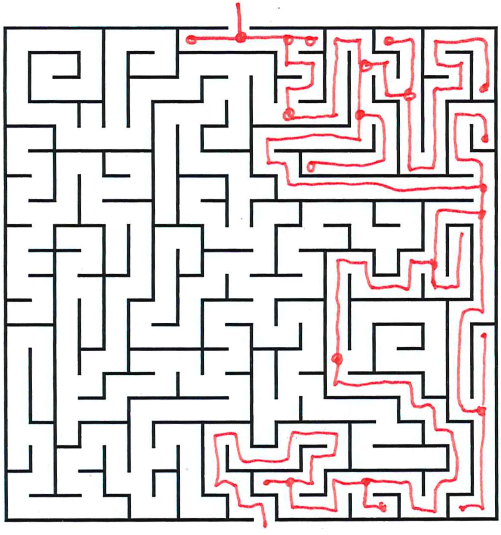
Laufzeiten einfacher Operationen

Operation	Matrix	Liste
Nachbarn/Nachfolger von $v \in V$ finden	$\Theta(n)$	$\Theta(\deg^+ v)$
$v \in V$ ohne Nachbar/Nachfolger finden	$\Theta(n^2)$	$\Theta(n)$
$(v,u) \in E$?	$\Theta(1)$	$\Theta(\deg^+ v)$
Kante einfügen	$\Theta(1)$	$\Theta(1)$
Kante (v,u) löschen	$\Theta(1)$	

Laufzeiten einfacher Operationen

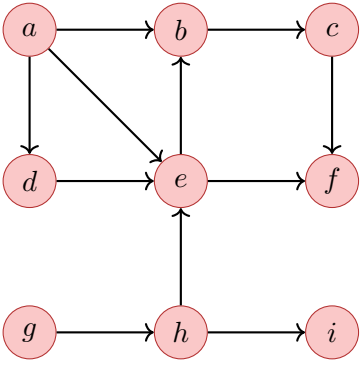
Operation	Matrix	Liste
Nachbarn/Nachfolger von $v \in V$ finden	$\Theta(n)$	$\Theta(\deg^+ v)$
$v \in V$ ohne Nachbar/Nachfolger finden	$\Theta(n^2)$	$\Theta(n)$
$(v,u) \in E$?	$\Theta(1)$	$\Theta(\deg^+ v)$
Kante einfügen	$\Theta(1)$	$\Theta(1)$
Kante (v,u) löschen	$\Theta(1)$	$\Theta(\deg^+ v)$

Tiefensuche

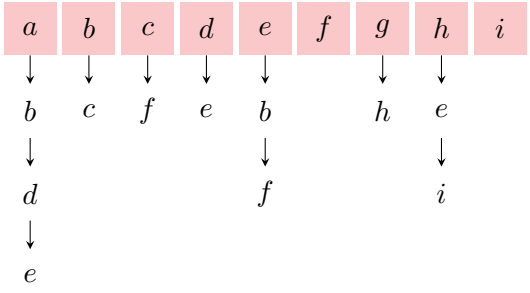


Graphen Traversieren: Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.

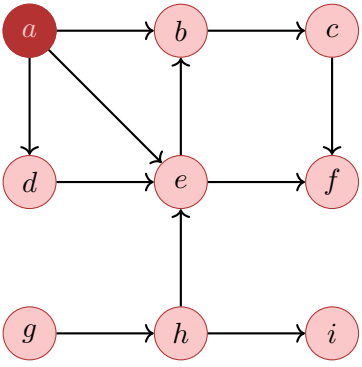


Adjazenzliste

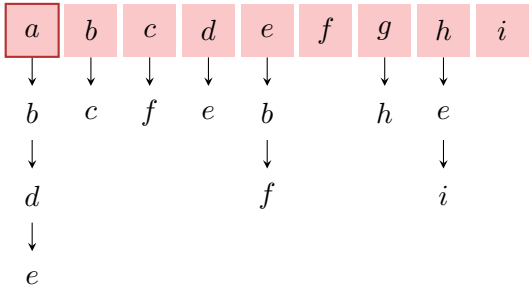


Graphen Traversieren: Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.

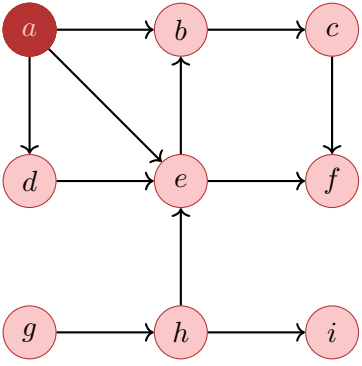


Adjazenzliste

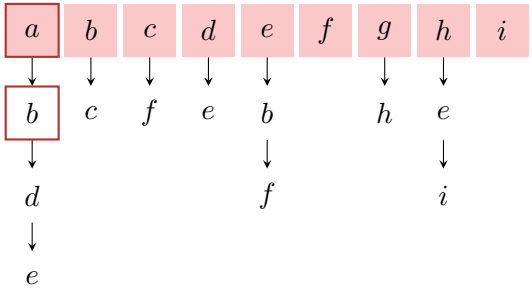


Graphen Traversieren: Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.

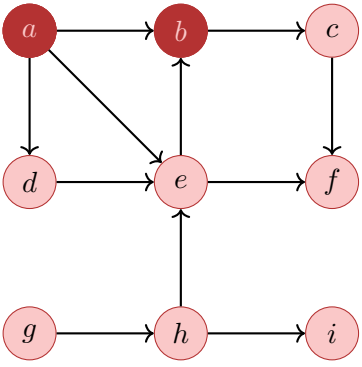


Adjazenzliste

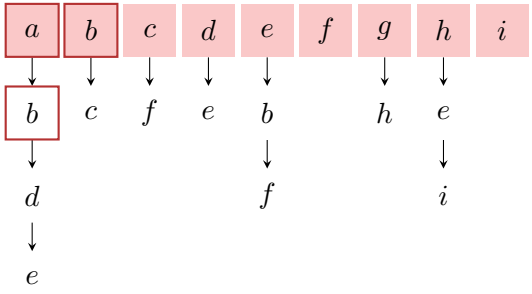


Graphen Traversieren: Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.

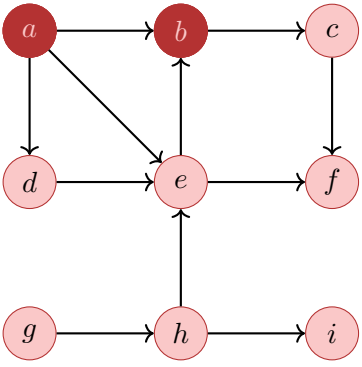


Adjazenzliste

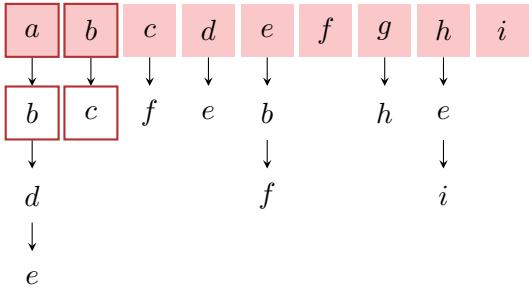


Graphen Traversieren: Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.

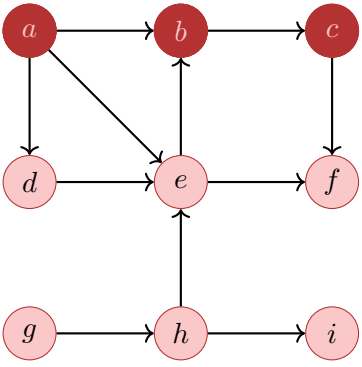


Adjazenzliste

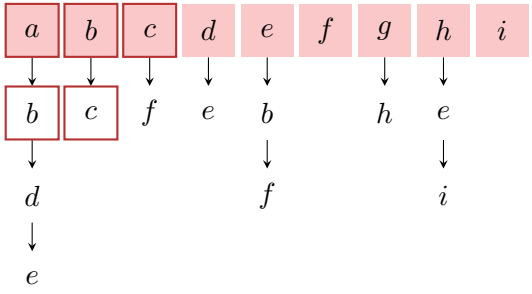


Graphen Traversieren: Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.

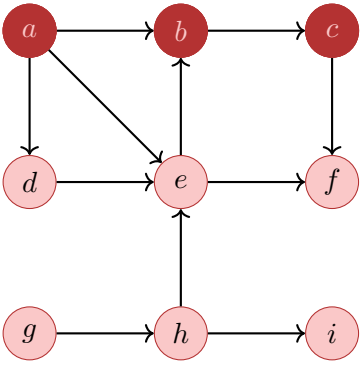


Adjazenzliste

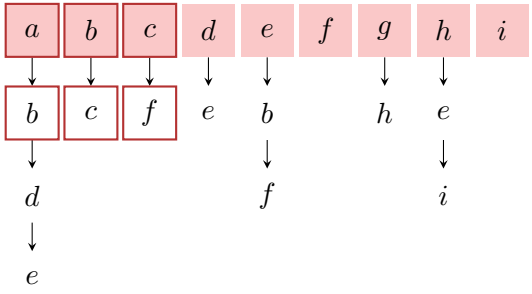


Graphen Traversieren: Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.

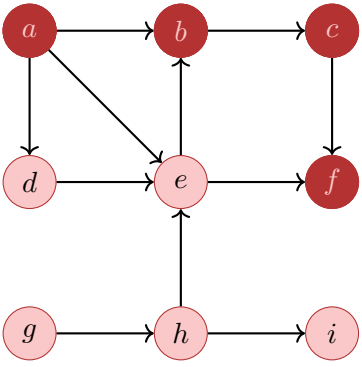


Adjazenzliste

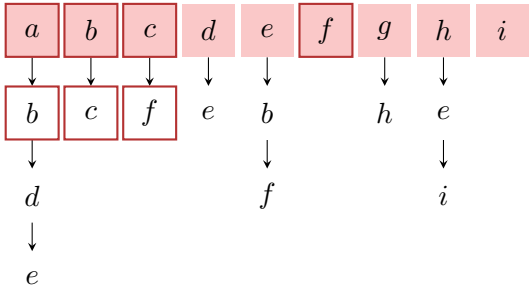


Graphen Traversieren: Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.

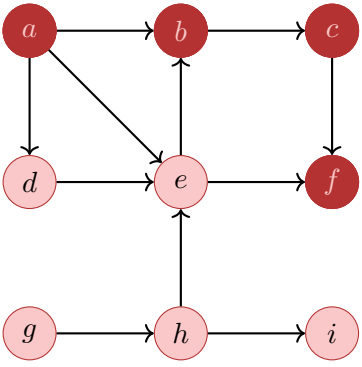


Adjazenzliste

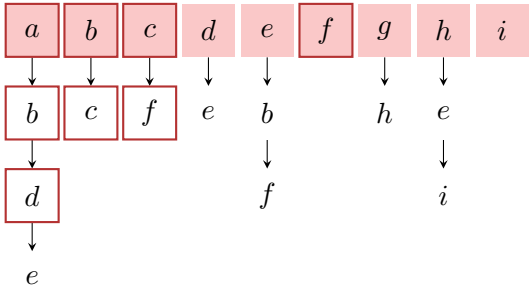


Graphen Traversieren: Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.

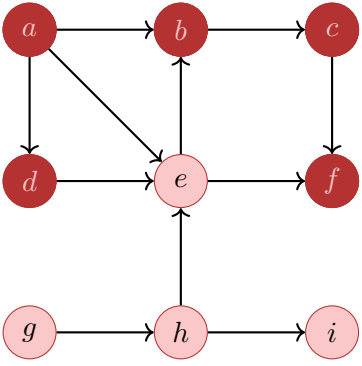


Adjazenzliste

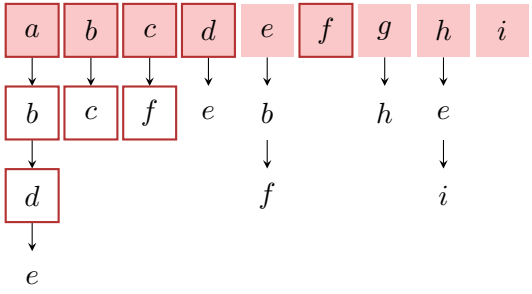


Graphen Traversieren: Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.

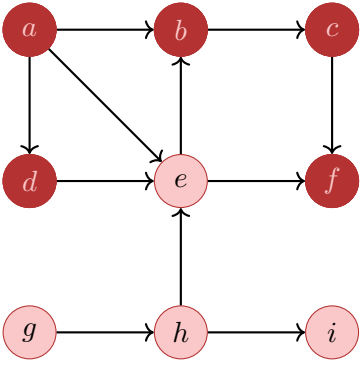


Adjazenzliste

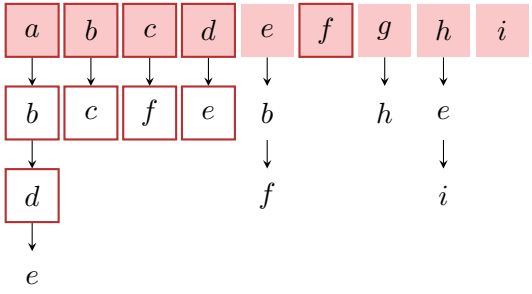


Graphen Traversieren: Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.

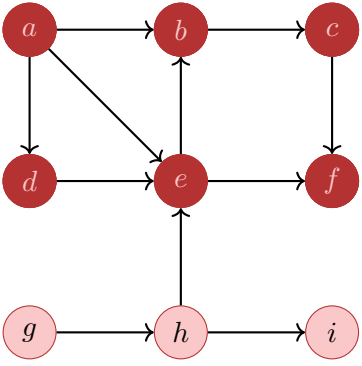


Adjazenzliste

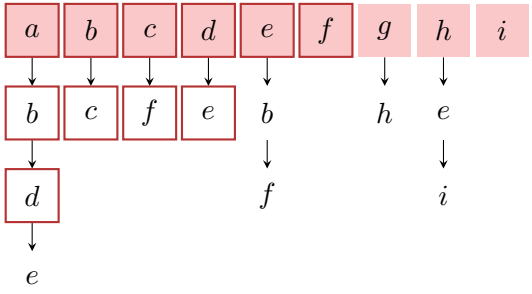


Graphen Traversieren: Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.

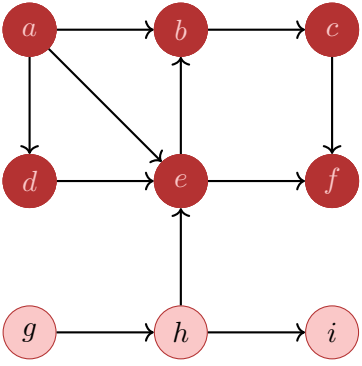


Adjazenzliste

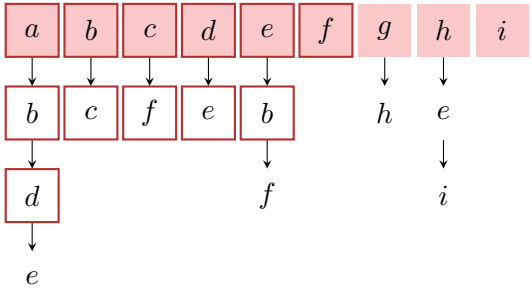


Graphen Traversieren: Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.

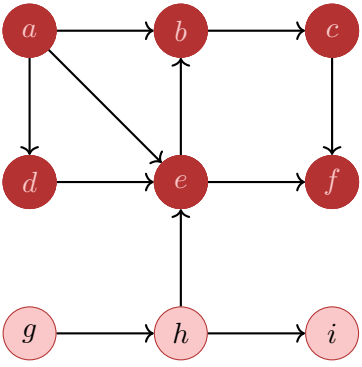


Adjazenzliste

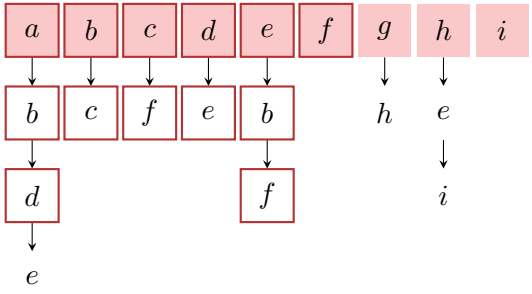


Graphen Traversieren: Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.

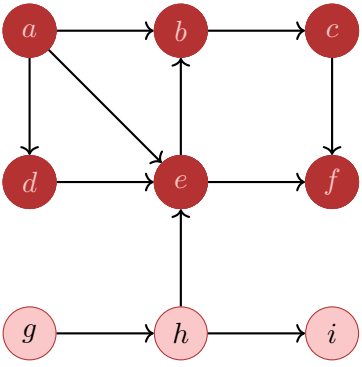


Adjazenzliste

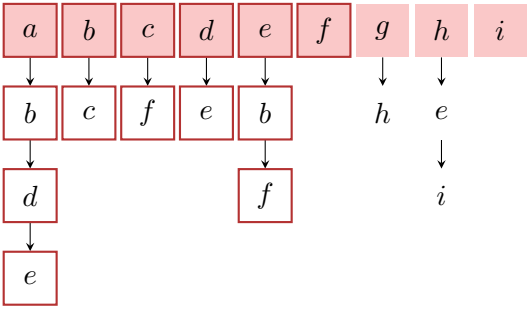


Graphen Traversieren: Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.

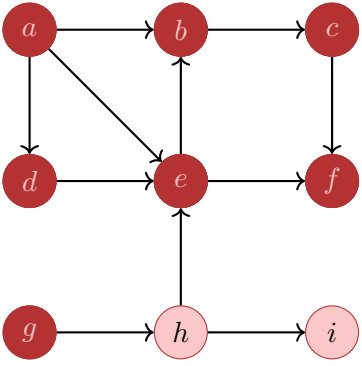


Adjazenzliste

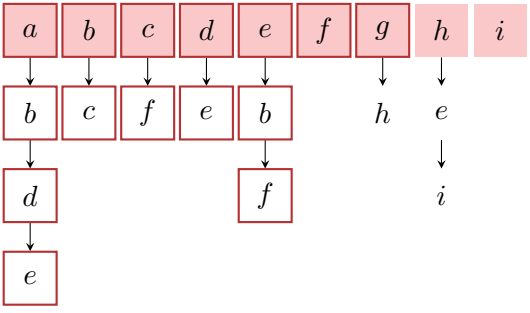


Graphen Traversieren: Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.

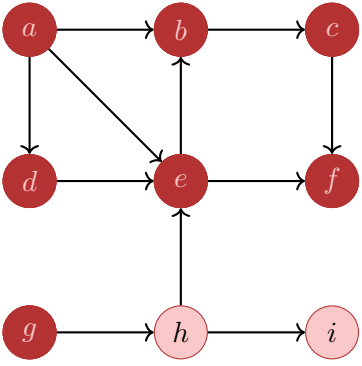


Adjazenzliste

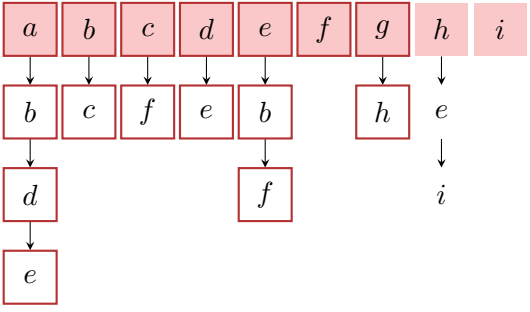


Graphen Traversieren: Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.

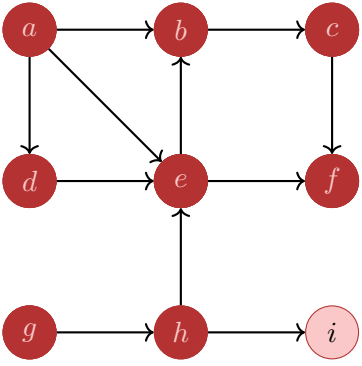


Adjazenzliste

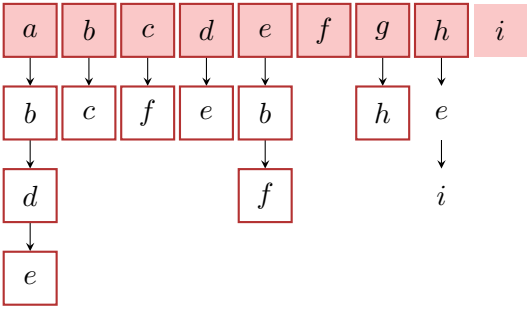


Graphen Traversieren: Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.

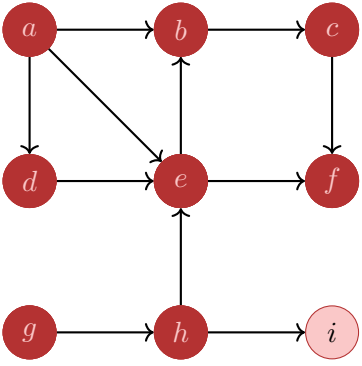


Adjazenzliste

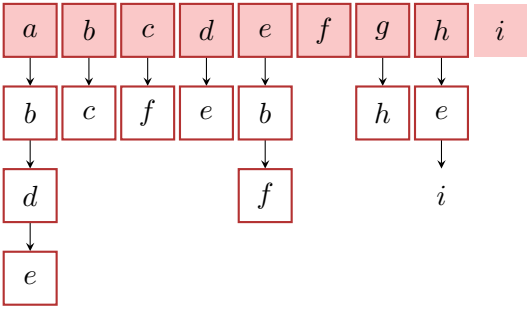


Graphen Traversieren: Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.

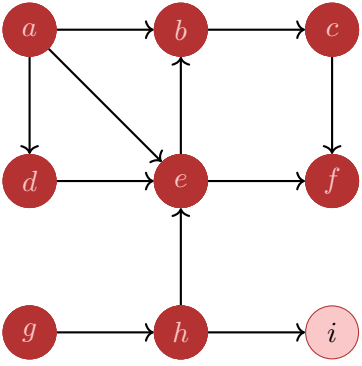


Adjazenzliste

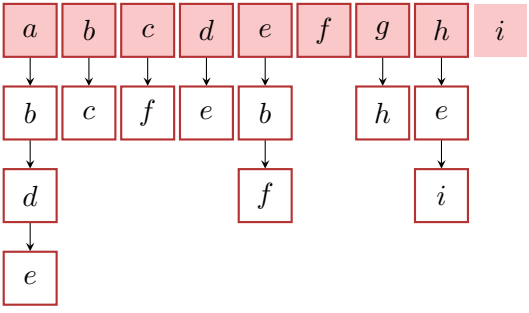


Graphen Traversieren: Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.

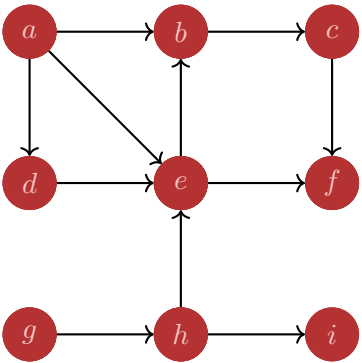


Adjazenzliste



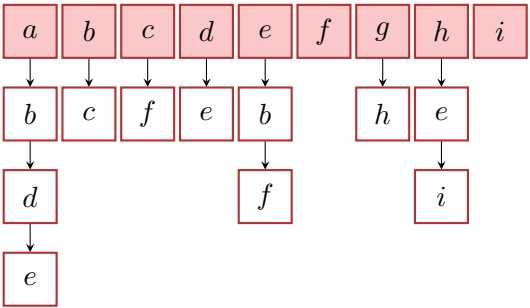
Graphen Traversieren: Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.



Reihenfolge $a, b, c, f, d, e, g, h, i$

Adjazenzliste



Farben

Konzeptuelle Färbung der Knoten

- **Weiss:** Knoten wurde noch nicht entdeckt.
- **Grau:** Knoten wurde entdeckt und zur Traversierung vorgemerkt / in Bearbeitung.
- **Schwarz:** Knoten wurde entdeckt und vollständig bearbeitet

Algorithmus Tiefensuche DFS-Visit(G, v)

Input: Graph $G = (V, E)$, Knoten v.

$v.color \leftarrow$ grey
foreach $w \in N^+(v)$ **do**
    if $w.color =$ white **then**
        DFS-Visit(G, w)
$v.color \leftarrow$ black

Tiefensuche ab Knoten v. Laufzeit (ohne Rekursion): $\Theta(\deg^+ v)$

Algorithmus Tiefensuche DFS-Visit(G)

Input: Graph $G = (V, E)$

foreach $v \in V$ **do**
    $v.color \leftarrow$ white

foreach $v \in V$ **do**
    if $v.color =$ white **then**
        DFS-Visit(G,v)

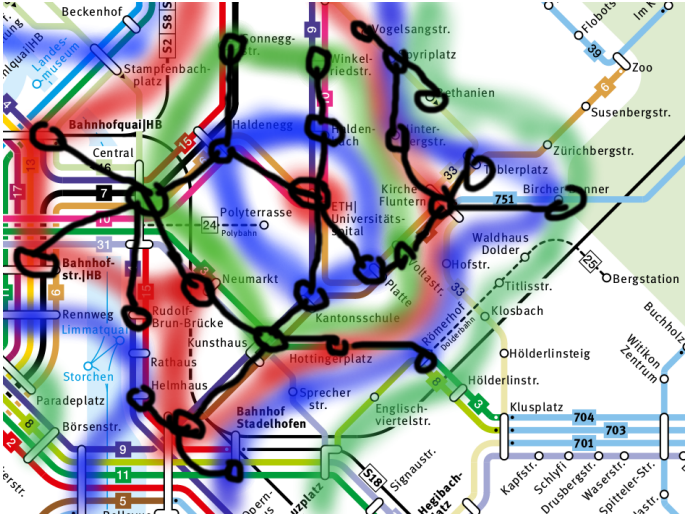
Tiefensuche für alle Knoten eines Graphen. Laufzeit
$\Theta(|V| + \sum_{v \in V}(\deg^+(v) + 1)) = \Theta(|V| + |E|)$.

Interpretation der Farben

Beim Traversieren des Graphen wird ein Baum (oder Wald) aufgebaut. Beim Entdecken von Knoten gibt es drei Fälle

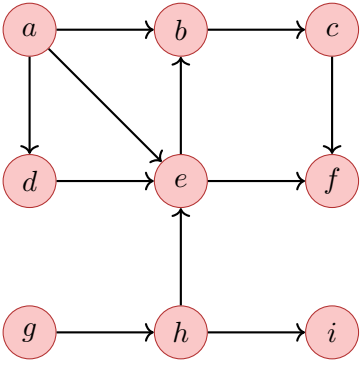
- Weisser Knoten: neue Baumkante
- Grauer Knoten: Zyklus („Rückwärtskante“)
- Schwarzer Knoten: Vorwärts-/Seitwärtskante

Breitensuche

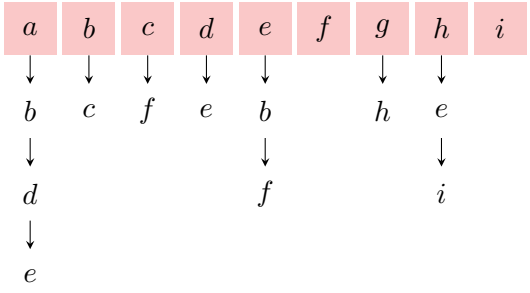


Graphen Traversieren: Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.

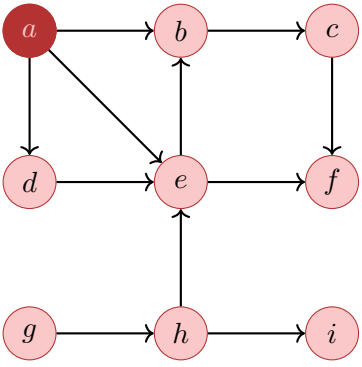


Adjazenzliste

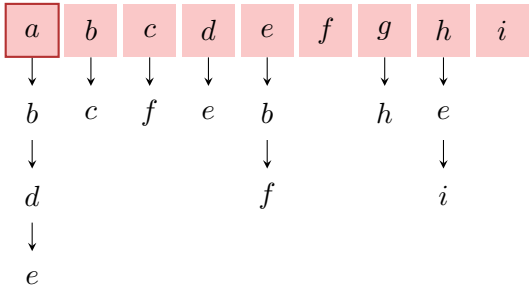


Graphen Traversieren: Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.

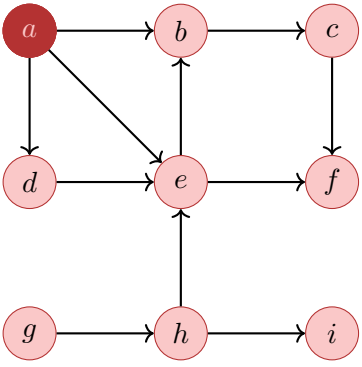


Adjazenzliste

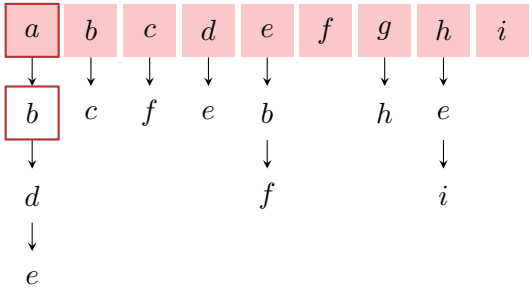


Graphen Traversieren: Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.

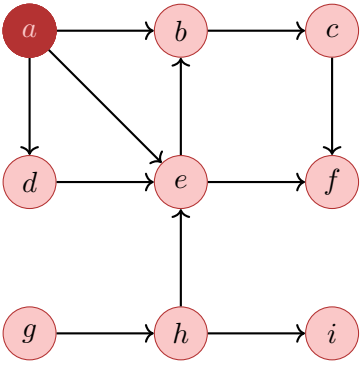


Adjazenzliste

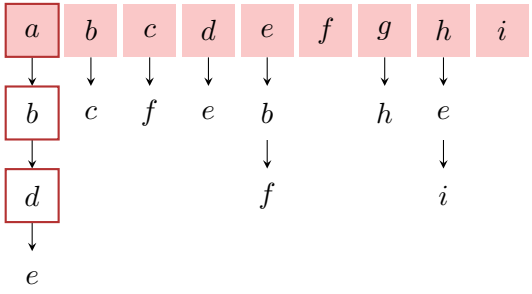


Graphen Traversieren: Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.

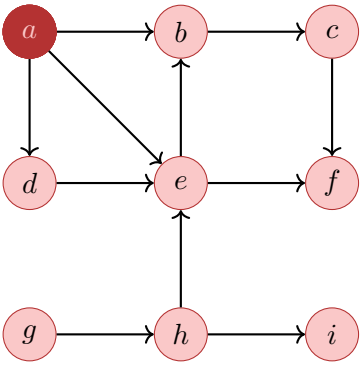


Adjazenzliste

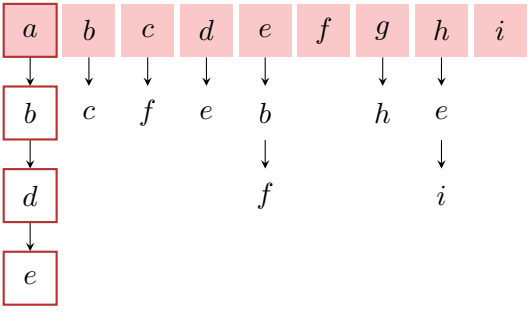


Graphen Traversieren: Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.

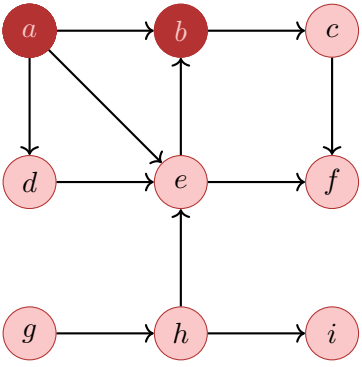


Adjazenzliste

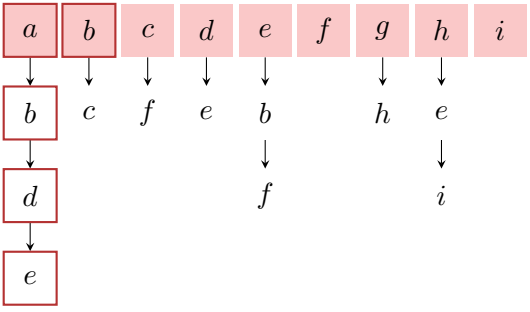


Graphen Traversieren: Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.

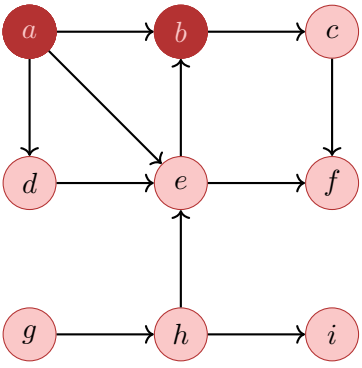


Adjazenzliste

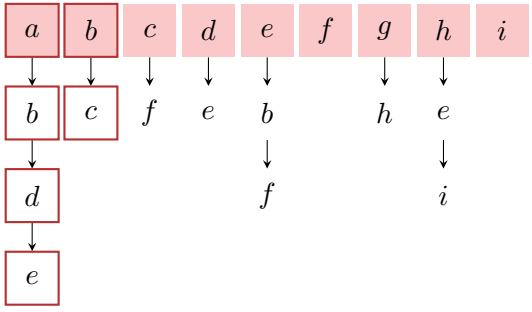


Graphen Traversieren: Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.

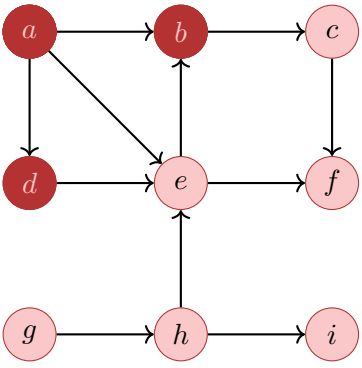


Adjazenzliste

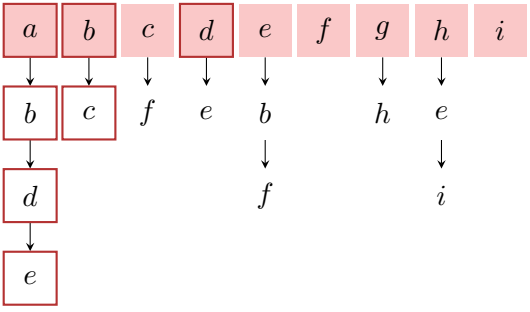


Graphen Traversieren: Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.

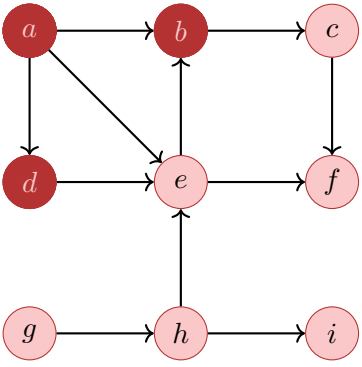


Adjazenzliste

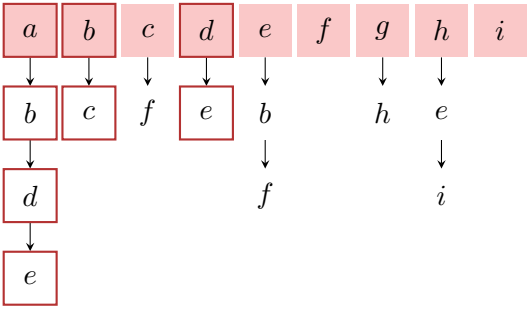


Graphen Traversieren: Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.

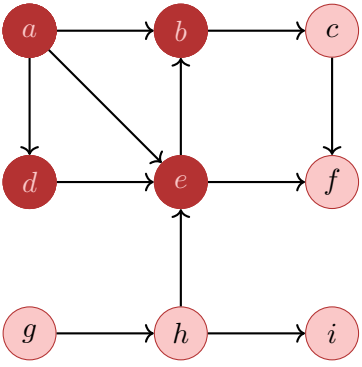


Adjazenzliste

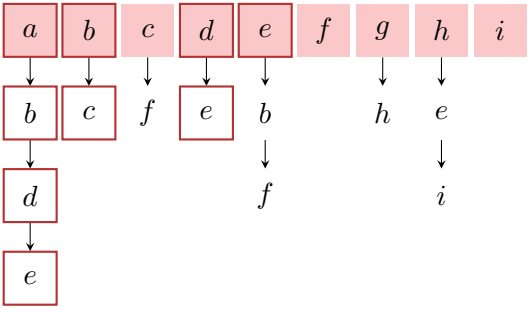


Graphen Traversieren: Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.

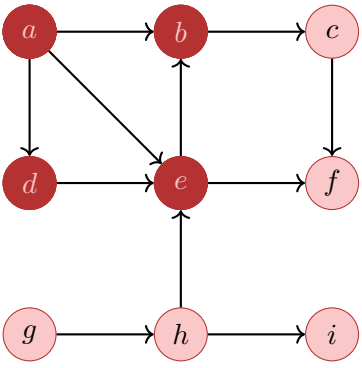


Adjazenzliste

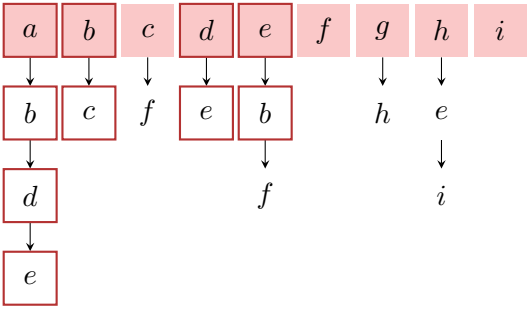


Graphen Traversieren: Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.

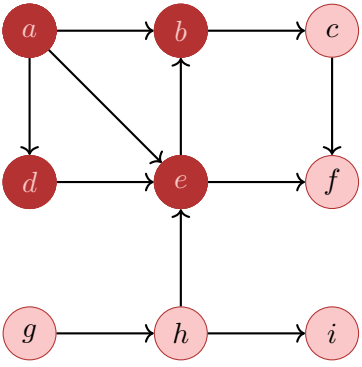


Adjazenzliste

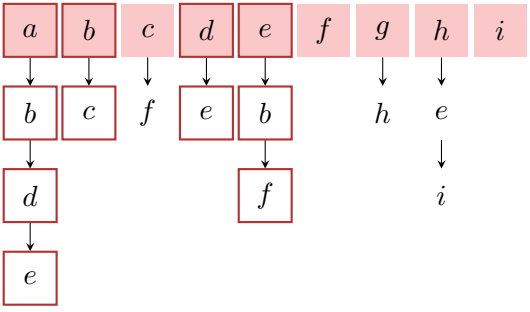


Graphen Traversieren: Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.

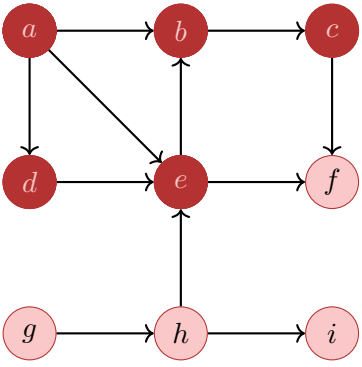


Adjazenzliste

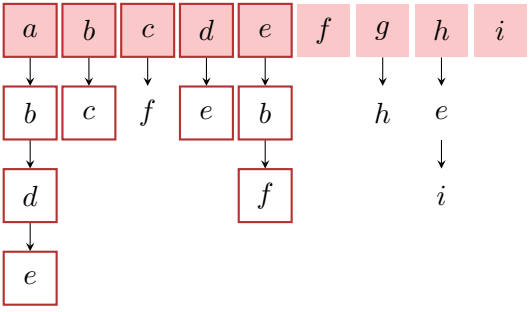


Graphen Traversieren: Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.

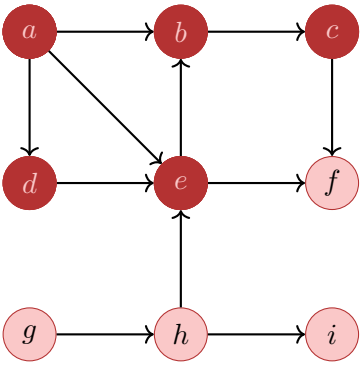


Adjazenzliste

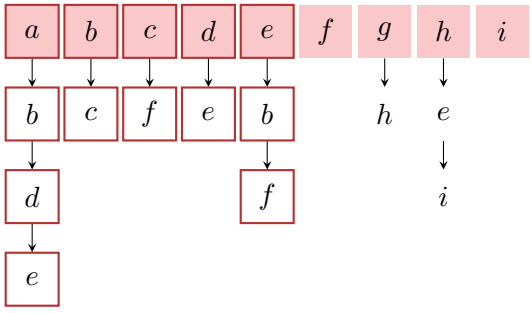


Graphen Traversieren: Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.

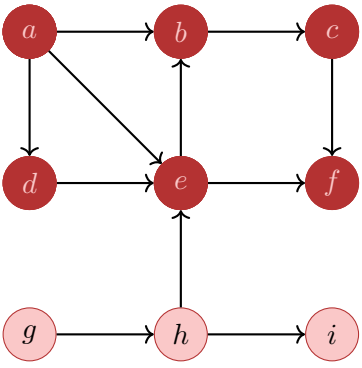


Adjazenzliste

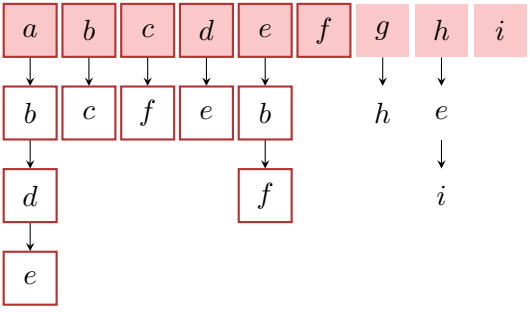


Graphen Traversieren: Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.

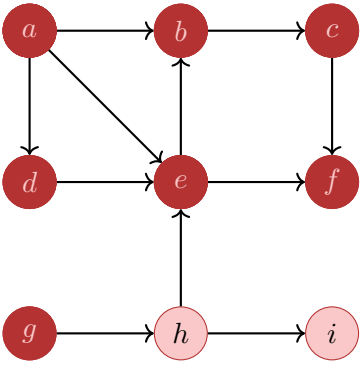


Adjazenzliste

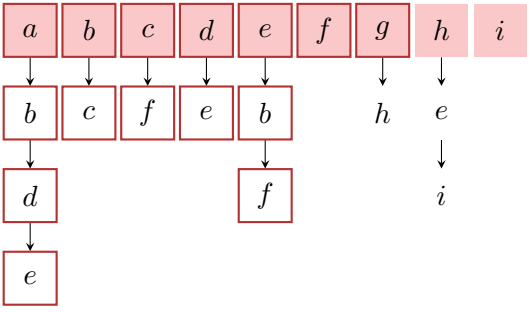


Graphen Traversieren: Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.

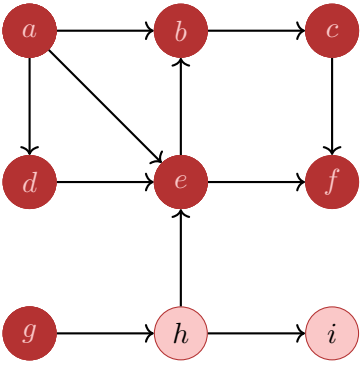


Adjazenzliste

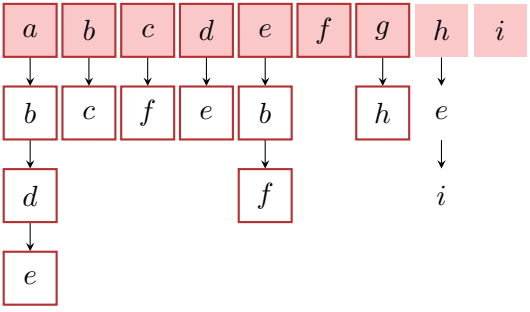


Graphen Traversieren: Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.

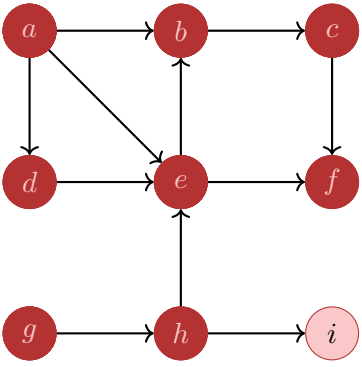


Adjazenzliste

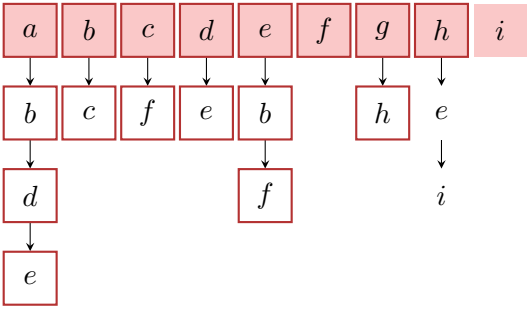


Graphen Traversieren: Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.

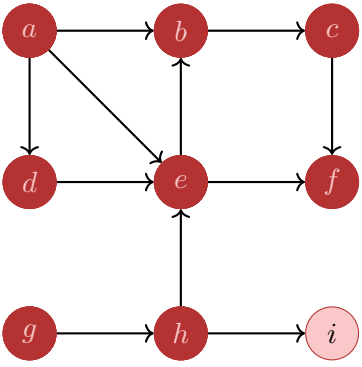


Adjazenzliste

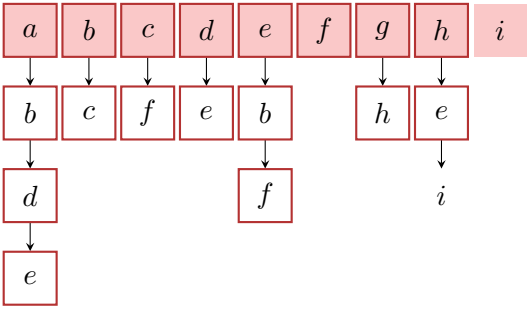


Graphen Traversieren: Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.

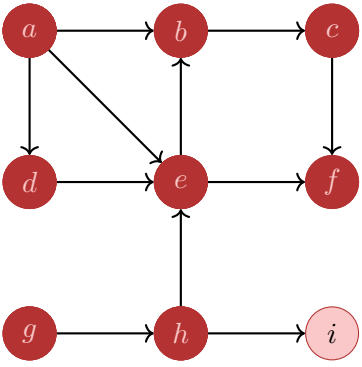


Adjazenzliste

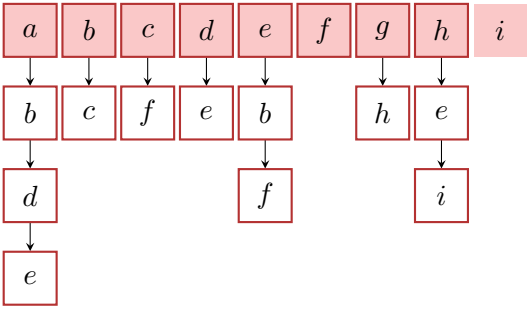


Graphen Traversieren: Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.

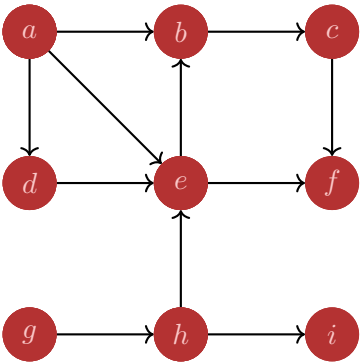


Adjazenzliste



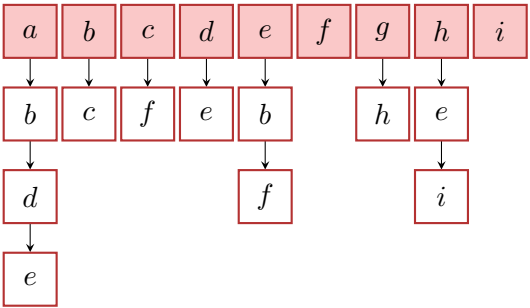
Graphen Traversieren: Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.



Reihenfolge $a, b, d, e, c, f, g, h, i$

Adjazenzliste



(Iteratives) BFS-Visit(G, v)

Input: Graph $G = (V, E)$

```
Queue Q ← ∅
v.color ← grey
enqueue(Q, v)
while Q ≠ ∅ do
    w ← dequeue(Q)
    foreach c ∈ N⁺(w) do
        if c.color = white then
            c.color ← grey
            enqueue(Q, c)
    w.color ← black
```

Algorithmus kommt mit $\mathcal{O}(|V|)$ Extraplatz aus.

Rahmenprogramm BFS-Visit(G)

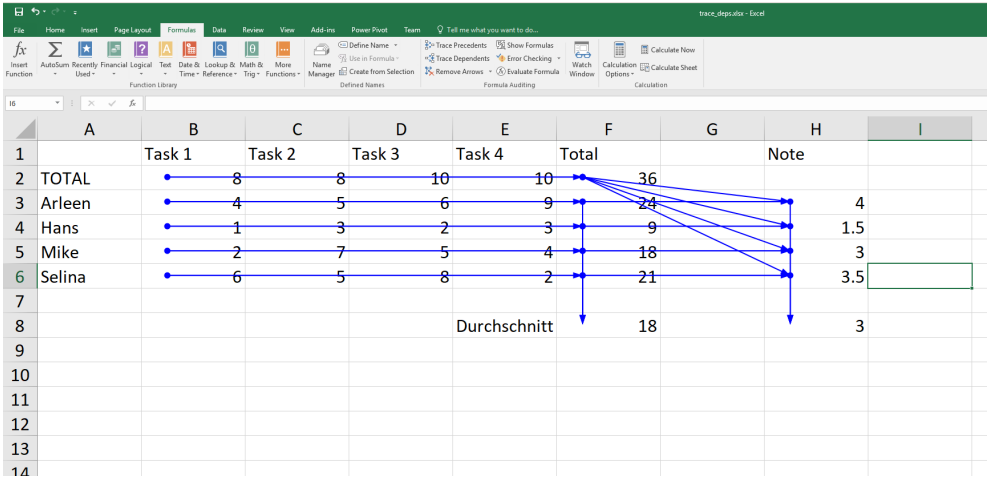
Input: Graph $G = (V, E)$

foreach $v \in V$ **do**
    $v.color \leftarrow$ white

foreach $v \in V$ **do**
    if $v.color =$ white **then**
        BFS-Visit(G,v)

Breitensuche für alle Knoten eines Graphen. Laufzeit $\Theta(|V| + |E|)$.

Topologisches Sortieren



Auswertungsreihenfolge?

Topologische Sortierung

Topologische Sortierung eines azyklischen gerichteten Graphen $G = (V, E)$:

Bijektive Abbildung

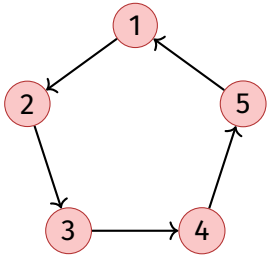
$$\mathrm{ord} : V \to \{1, \ldots, |V|\}$$

so dass

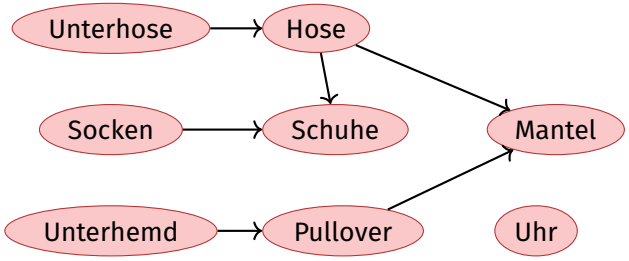
$$\mathrm{ord}(v) < \mathrm{ord}(w) \; \forall \; (v, w) \in E.$$

Identifizieren Wert i mit dem Element $v_i := \mathrm{ord}^{-1}(i)$. Topologische Sortierung $\widehat{=} \langle v_1, \ldots, v_{|V|} \rangle$.

(Gegen-)Beispiele



Zyklischer Graph: kann nicht topologisch sortiert werden.



Eine mögliche topologische Sortierung des Graphen:
Unterhemd, Pullover, Unterhose,Uhr, Hose, Mantel, Socken, Schuhe

Beobachtung

Theorem 20

Ein gerichteter Graph $G = (V, E)$ besitzt genau dann eine topologische Sortierung, wenn er kreisfrei ist

Algorithmus Topological-Sort(G)

Input: Graph $G = (V, E)$.
Output: Topologische Sortierung ord

Stack $S \leftarrow \emptyset$
foreach $v \in V$ **do** $A[v] \leftarrow 0$
foreach $(v, w) \in E$ **do** $A[w] \leftarrow A[w] + 1$ // Eingangsgrade berechnen
foreach $v \in V$ with $A[v] = 0$ **do** push(S, v) // Merke Nodes mit Eingangsgrad 0
$i \leftarrow 1$
while $S \neq \emptyset$ **do**
    $v \leftarrow$ pop(S); $\mathrm{ord}[v] \leftarrow i$; $i \leftarrow i + 1$ // Wähle Knoten mit Eingangsgrad 0
    foreach $(v, w) \in E$ **do** // Verringere Eingangsgrad der Nachfolger
        $A[w] \leftarrow A[w] - 1$
        if $A[w] = 0$ **then** push(S, w)

if $i = |V| + 1$ **then return** ord **else return** "Cycle Detected"

Algorithmus Korrektheit

Theorem 21

Sei $G = (V, E)$ ein gerichteter, kreisfreier Graph. Der Algorithmus `TopologicalSort`(G) *berechnet in Zeit $\Theta(|V| + |E|)$ eine topologische Sortierung ord für G.*

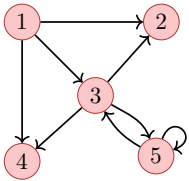
Algorithmus Korrektheit

Theorem 22

Sei $G = (V, E)$ ein gerichteter, nicht kreisfreier Graph. Der Algorithmus **`TopologicalSort`**(G) *terminiert in Zeit $\Theta(|V|+|E|)$ und detektiert Zyklus.*

Adjazenzmatrizen multipliziert

$$B := A_G^2 = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix}^2 = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 2 \end{pmatrix}$$



Interpretation

Theorem 23

Sei $G = (V, E)$ ein Graph und $k \in \mathbb{N}$. Dann gibt das Element $a_{i,j}^{(k)}$ der Matrix $(a_{i,j}^{(k)})_{1 \leq i,j \leq n} = (A_G)^k$ die Anzahl der Wege mit Länge k von v_i nach v_j an.

Beweis

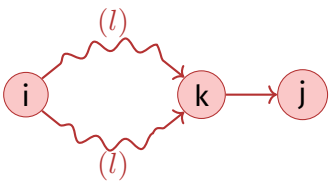
Per Induktion.

Anfang: Klar für $k = 1$. $a_{i,j} = a_{i,j}^{(1)}$.

Hypothese: Aussage wahr für alle $k \leq l$

Schritt ($l \to l + 1$):

$$a_{i,j}^{(l+1)} = \sum_{k=1}^{n} a_{i,k}^{(l)} \cdot a_{k,j}$$



$a_{k,j} = 1$ g.d.w. Kante von k nach j, 0 sonst. Summe zählt die Anzahl Wege der Länge l vom Knoten v_i zu allen Knoten v_k welche direkte Verbindung zu Knoten v_j haben, also alle Wege der Länge $l + 1$.

Relation

Gegeben: endliche Menge V

(Binäre) **Relation** R auf V: Teilmenge des kartesischen Produkts $V \times V = \{(a, b) | a \in V, b \in V\}$

Relation $R \subseteq V \times V$ heisst

- reflexiv, wenn $(v, v) \in R$ für alle $v \in V$
- symmetrisch, wenn $(v, w) \in R \Rightarrow (w, v) \in R$
- transitiv, wenn $(v, x) \in R$, $(x, w) \in R \Rightarrow (v, w) \in R$

Die (Reflexive) Transitive Hülle R^* von R ist die kleinste Erweiterung $R \subseteq R^* \subseteq V \times V$ von R, so dass R^* reflexiv und transitiv ist.

Graphen und Relationen

Graph $G = (V, E)$
Adjazenzen $A_G \mathrel{\widehat{=}}$ Relation $E \subseteq V \times V$ auf V

Graphen und Relationen

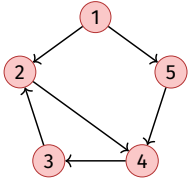
Graph $G = (V, E)$
Adjazenzen $A_G \mathrel{\widehat{=}}$ Relation $E \subseteq V \times V$ auf V

- reflexiv $\Leftrightarrow a_{i,i} = 1$ für alle $i = 1, \ldots, n$. (Schleifen)
- symmetrisch $\Leftrightarrow a_{i,j} = a_{j,i}$ für alle $i, j = 1, \ldots, n$ (ungerichtet)
- transitiv $\Leftrightarrow$ $(u, v) \in E$, $(v, w) \in E \Rightarrow (u, w) \in E$. (Erreichbarkeit)

Reflexive Transitive Hülle

Reflexive transitive Hülle von G $\Leftrightarrow$ Erreichbarkeitsrelation E^*: $(v, w) \in E^*$ gdw. $\exists$ Weg von Knoten v zu w.

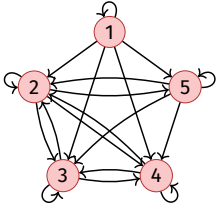
$$\begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$



$G = (V, E)$

$\Rightarrow$

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$



$G^* = (V, E^*)$

Algorithmus $A \cdot A$

Input: (Adjazenz-)Matrix $A = (a_{ij})_{i,j=1\ldots n}$
Output: Matrixprodukt $B = (b_{ij})_{i,j=1\ldots n} = A \cdot A$

$B \leftarrow 0$
for $r \leftarrow 1$ **to** n **do**
  for $c \leftarrow 1$ **to** n **do**
    for $k \leftarrow 1$ **to** n **do**
      $b_{rc} \leftarrow b_{rc} + a_{rk} \cdot a_{kc}$ // Anzahl Pfade
return B

Berechnet Anzahl Pfade der Länge 2

Algorithmus $A \otimes A$

Input: Adjazenzmatrix $A = (a_{ij})_{i,j=1\ldots n}$
Output: Modifiziertes Matrixprodukt $B = (b_{ij})_{i,j=1\ldots n} = A \otimes A$

$B \leftarrow A$ // Pfade erhalten
for $r \leftarrow 1$ **to** n **do**
  for $c \leftarrow 1$ **to** n **do**
    for $k \leftarrow 1$ **to** n **do**
      $b_{rc} \leftarrow \max\{b_{rc}, a_{rk} \cdot a_{kc}\}$ // Pfad: ja/nein
return B

Berechnet Existenz von Pfaden der Längen 1 und 2

Berechnung Reflexive Transitive Hülle

Ziel: Berechnung von $B = (b_{ij})_{1 \leq i,j \leq n}$ mit $b_{ij} = 1 \Leftrightarrow (v_i, v_j) \in E^*$

Berechnung Reflexive Transitive Hülle

Ziel: Berechnung von $B = (b_{ij})_{1 \leq i,j \leq n}$ mit $b_{ij} = 1 \Leftrightarrow (v_i, v_j) \in E^*$ Erste Idee:

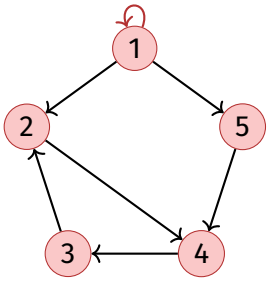
- Starte mit $B \leftarrow A$ und setze $b_{ii} = 1$ für alle i (Reflexivität).
- Berechne

$$B_n = \bigotimes_{i=1}^{n} B$$

mit Potenzen von 2 $B_2 := B \otimes B$, $B_4 := B_2 \otimes B_2$, $B_8 = B_4 \otimes B_4$ ...
$\Rightarrow$ Laufzeit $n^3 \lceil \log_2 n \rceil$

Verbesserung: Algorithmus von Warshall (1962)

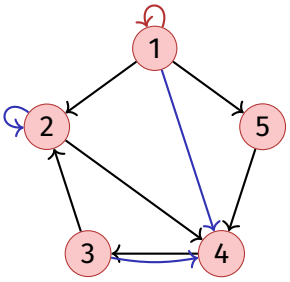
Induktiver Ansatz: Alle Wege bekannt über Knoten aus $\{v_i : i < k\}$.
Hinzunahme des Knotens v_k.



$$\begin{bmatrix} 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Verbesserung: Algorithmus von Warshall (1962)

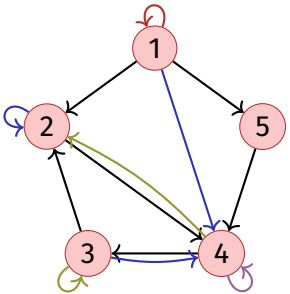
Induktiver Ansatz: Alle Wege bekannt über Knoten aus $\{v_i : i < k\}$.
Hinzunahme des Knotens v_k.



$$\begin{bmatrix} 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Verbesserung: Algorithmus von Warshall (1962)

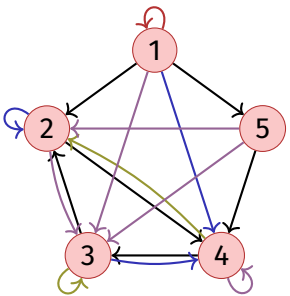
Induktiver Ansatz: Alle Wege bekannt über Knoten aus $\{v_i : i < k\}$.
Hinzunahme des Knotens v_k.



$$\begin{bmatrix} 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Verbesserung: Algorithmus von Warshall (1962)

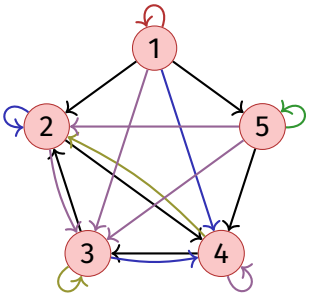
Induktiver Ansatz: Alle Wege bekannt über Knoten aus $\{v_i : i < k\}$.
Hinzunahme des Knotens v_k.



$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

Verbesserung: Algorithmus von Warshall (1962)

Induktiver Ansatz: Alle Wege bekannt über Knoten aus $\{v_i : i < k\}$.
Hinzunahme des Knotens v_k.



$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Algorithmus TransitiveClosure(A_G)

Input: Adjazenzmatrix $A_G = (a_{ij})_{i,j=1\ldots n}$
Output: Reflexive Transitive Hülle $B = (b_{ij})_{i,j=1\ldots n}$ von G

$B \leftarrow A_G$
for $k \leftarrow 1$ **to** n **do**
    $b_{kk} \leftarrow 1$ // Reflexivität
    for $r \leftarrow 1$ **to** n **do**
        for $c \leftarrow 1$ **to** n **do**
            $b_{rc} \leftarrow \max\{b_{rc}, b_{rk} \cdot b_{kc}\}$ // Alle Wege über v_k
return B

Laufzeit des Algorithmus $\Theta(n^3)$.

Korrektheit des Algorithmus (Induktion)

Invariante (k): alle Wege über Knoten mit maximalem Index $< k$ berücksichtigt

- **Anfang ($k = 1$)**: Alle direkten Wege (alle Kanten) in A_G berücksichtigt.
- **Hypothese**: Invariante (k) erfüllt.
- **Schritt** ($k \to k + 1$): Für jeden Weg von v_i nach v_j über Knoten mit maximalen Index k: nach Hypothese $b_{ik} = 1$ und $b_{kj} = 1$. Somit im k-ten Schleifendurchlauf: $b_{ij} \leftarrow 1$.