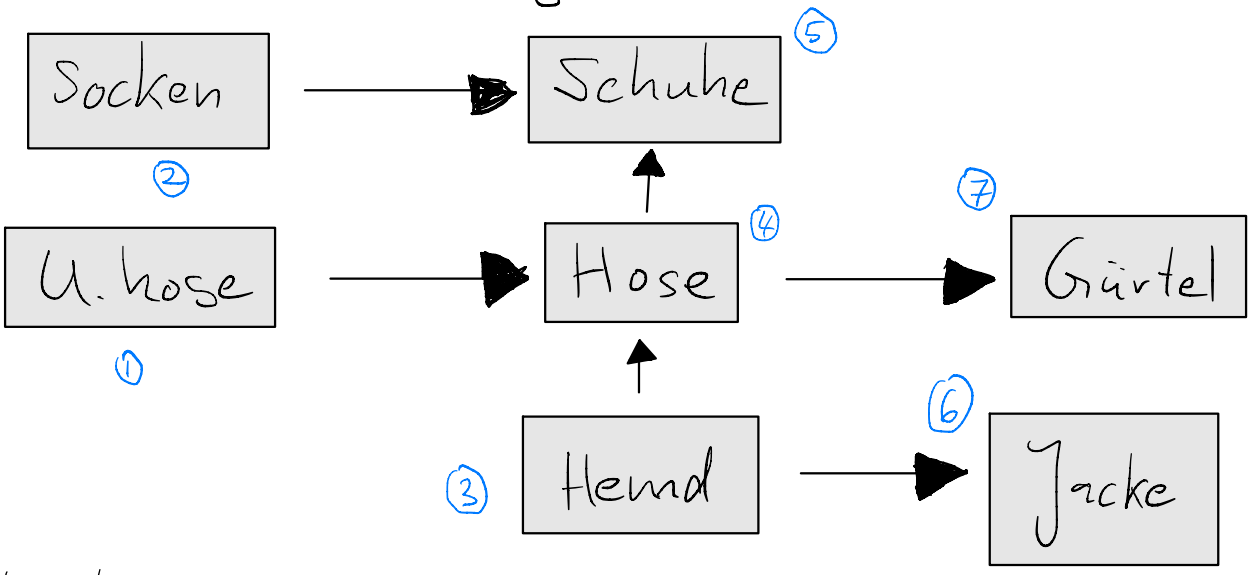
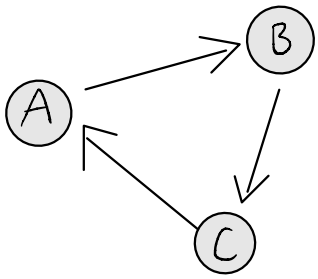
Abhängige Aufgaben planen

Beispiel: Kleidungsstücke anziehen



topologische Sortierung: Reihenfolge, die alle Abhängigkeiten erfüllt

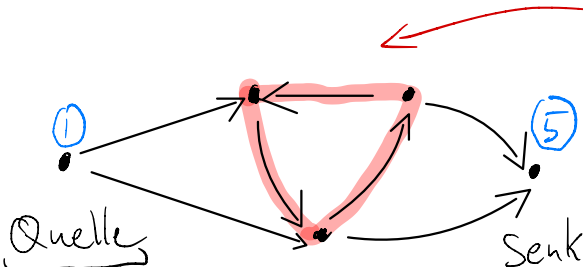
wann möglich?



jeder Knoten hat Nachfolger

⇝ keine topologische Sortierung möglich

(topologisch letzter Knoten darf keine Nachfolger haben)



gerichteter Zyklus

⇝ keine topo. Sort. möglich

Quelle kein Vorgänger

Senke kein Nachfolger

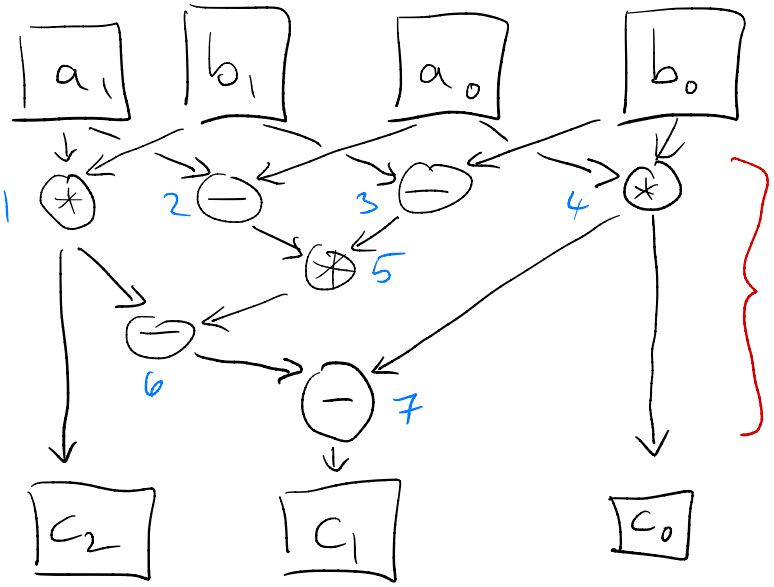
Behauptung: ∃ topo. Sort

$\iff \nexists$ ger. Zyklus

Beweis: "⇒" s. oben "⇐" per Algorithmus (später)

weiteres Bsp.: Reihenfolge für Berechnung

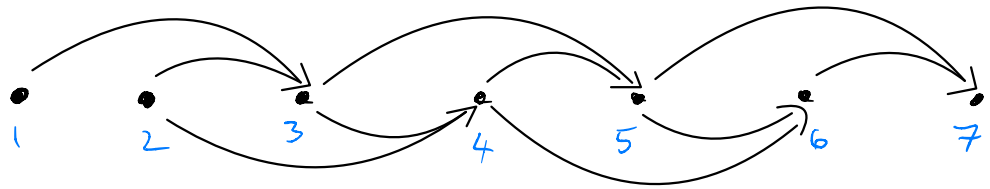
Karatsuba's Algorithmus



Rechenoperationen

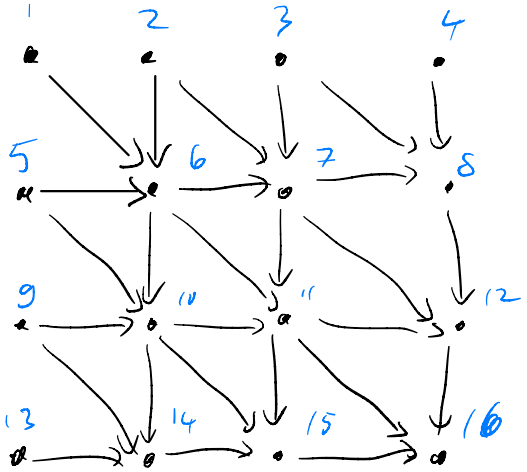
dynamische Programmierung

Fibonacci



$F_n = F_{n-1} + F_{n-2}$

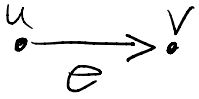
längste gemeinsame Teilfolge



$$T(i,j) = \max\{T(i-1,j),\ T(i,j-1),\ T(i-1,j-1) + c_{ij}\}$$

Definition: gerichteter Graph $G = (V, E)$

wie ungerichteter Graph, nur dass Kanten geordnete Paare sind



entspricht $e = (u, v) \in E$

Begriffe:

Vorgänger von v

Nachfolger von u

$\deg_{in}(u)$ Eingangsgrad

$\deg_{out}(u)$ Ausgangsgrad

Quelle u: $\deg_{in}(u) = 0$

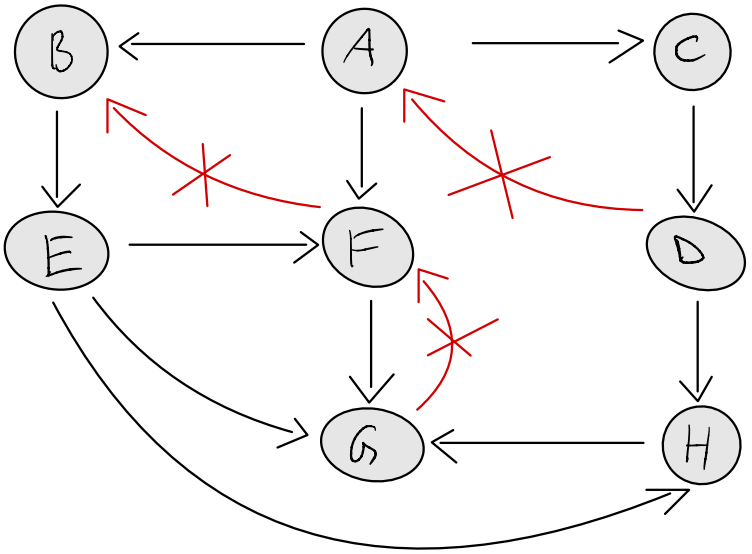
Senke v: $\deg_{out}(v) = 0$

gerichteter Weg: 

ger. Zyklus: $v_0 = v_e$

ger. Pfad: kein Knoten wiederholt

Topologisch Sortieren



Ansatz:

- finde Senke v
- setze v an letzte Stelle
- entferne v und löse Rest rekursiv

Wie?
Was wenn keine Senke?
mehrere S.?

werden zeigen (per Alg.):

$\nexists$ Zyklus $\Rightarrow$ $\exists$ Senke

Folglich: Ansatz funktioniert

(auch in rekursiven Instanzen keine Zyklen)

Pause

Path(u): (finde langen Pfad von u, unmarkiert)

markiere u

if ∃ Nachfolger v, unmarkiert

Path(v)

Beispiel: Path(A): A B E F G

Eigenschaften: (1) Path(u) markiert Pfad P mit Start u

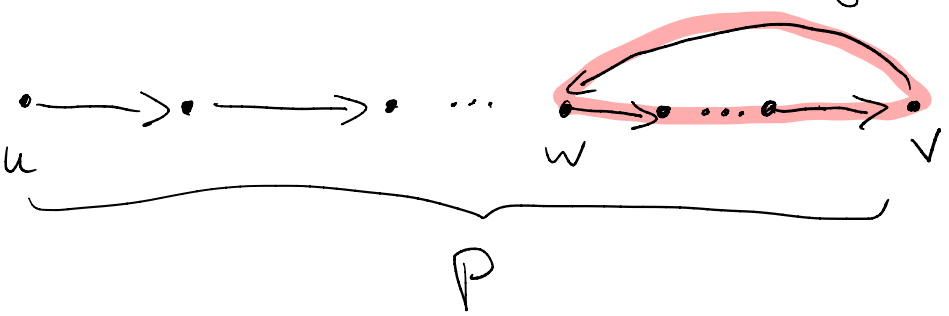
(2) Endknoten v von P hat alle Nachfolger markiert

Behauptung: ∄ ger. Zyklus ⟹ v ist Senke

Beweis (indirekt):

angenommen: v nicht Senke

E.(2) ⇝ v hat markierten Nachfolger w



⇝ ∃ ger. Zyklus ☒

Laufzeit:

Idee: anstatt Suche neu zu starten, Pfad wiederverwenden soweit möglich

Beispiel:

A B E F ~~G~~ 8

A B E ~~F~~ 7

A B E ~~H~~ 6

A B ~~E~~ 5

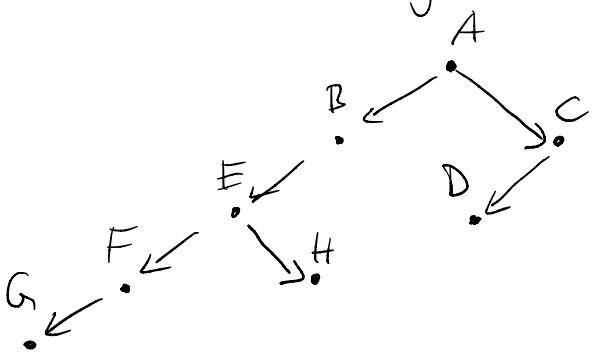
A ~~B~~ 4

A C ~~D~~ 3

A ~~C~~ 2

~~A~~ 1

Kompakte Darstellung:



rekursive Umsetzung

Visit (u):

markiere u

For Nachfolger v, unmarkiert:

| Visit (v)

Füge u zur top. Sort. hinzu

DFS (G): (Tiefensuche, Rahmenprogramm)

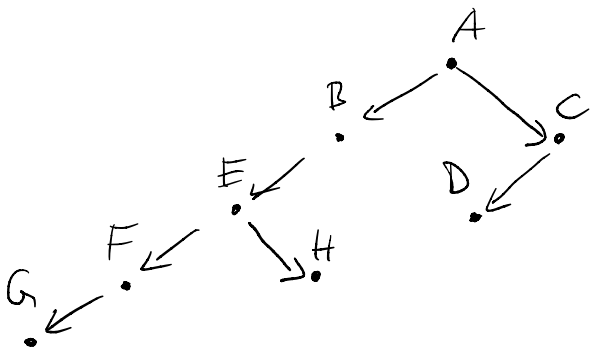
alle Knoten unmarkiert

For $u_0 \in V$, unmarkiert:

Visit (u_0)

Beispiel:

Rekursionsbaum wie Darstellung zuvor



Laufzeitanalyse

Adjazenzmatrix

Anzahl Visit Aufrufe: n (einen pro Knoten)

Zeit pro Visit Aufruf: $O(n)$ (Nachfolger durchlaufen)
(ohne Zeit in rek. Aufrufen)

Adjazenzliste

$Adj[u]$ = Liste der Nachfolger von u (beliebige Reihenfolge)

Zeit für Visit(u): $O(1 + deg_{out}(u))$

Total:

$$\sum_{u \in V} (1 + deg_{out}(u)) = |V| + \underbrace{\sum_{u \in V} deg_{out}(u)}_{= |E|}$$

$$= |V| + |E|$$

$\rightsquigarrow$ Tiefensuche hat Laufzeit $O(|V| + |E|)$

Pause

Tiefensuche: Tiefer verstehen

Start- und Endzeitpunkte der Visit-Aufrufe

Visit (u):

pre[u] ← T ; T ← T+1

... (wie zuvor)

post[u] ← T ; T ← T+1

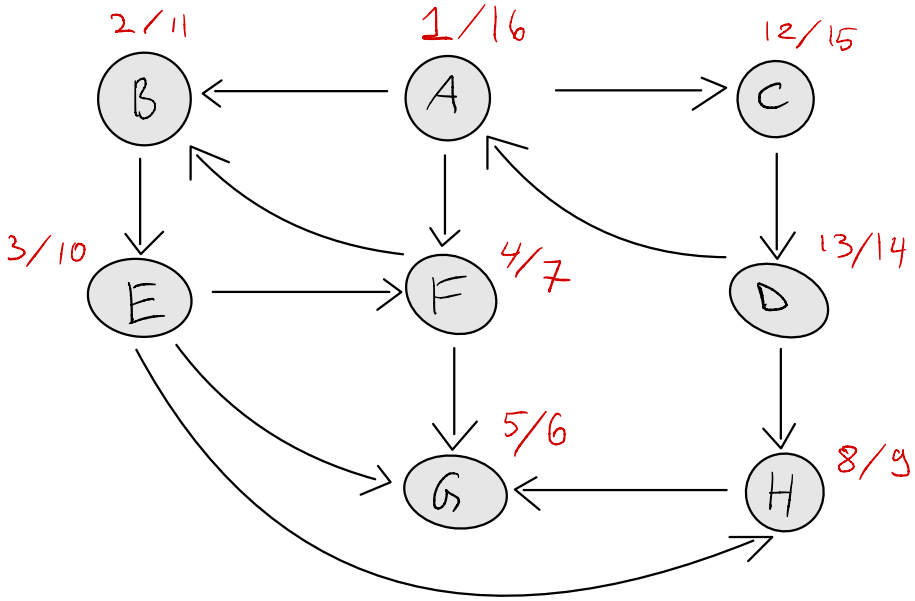
DFS(G):

T ← 1

... (wie zuvor)

Beispiel:

pre/post

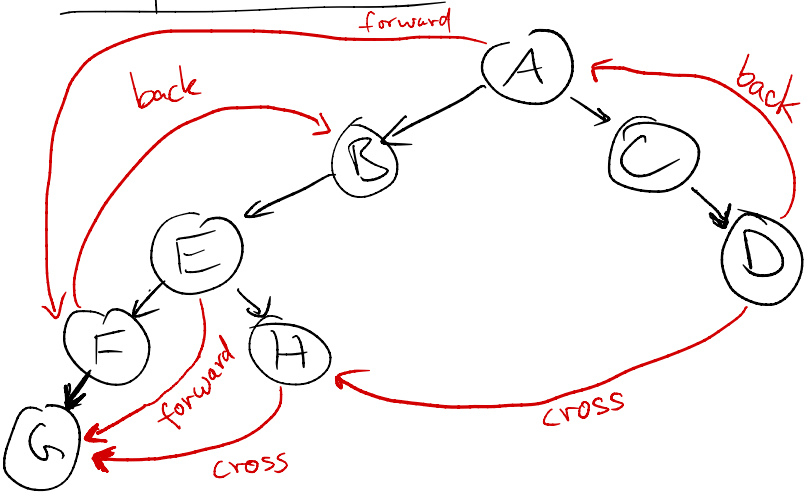


pre-order: A B E F G H C D

post-order: G F H E B D C A

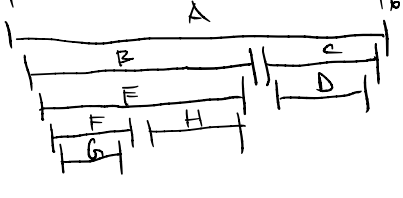
(umgekehrt topologisch wenn azyklisch)

Tiefensuchbaum



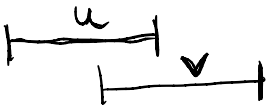
DFS-baum $\simeq$ Verschachtelung der Intervalle I_u

$$I_u = \{ pre[u], \ldots, post[u] \}$$



Klassifizierung der Kanten

Kante $(u, v) \in E$

I_u vs I_v	Klassifizierung
5. 	nicht möglich (Aufruf für v muss enden vor Aufruf für u)
3. 	forward oder DFS-Baum
4.  (w beliebig)	forward z.B. (A, F)
1. 	back z.B. (F, B)
2. 	cross z.B. (D, H)
6. 	nicht möglich (v noch nicht markiert als (u,v) betrachtet wurde)

Beobachtungen:

$\exists$ "back" Kante $\Rightarrow$ $\exists$ gerichteter Zyklus (1)

$(u,v) \in E$, nicht "back" $\Rightarrow$ $post[u] \geq post[v]$ (2)

(1), (2) $\rightsquigarrow$ $\nexists$ ger. Zyklus $\Leftrightarrow$ umgekehrte post order ist topologische Sortierung

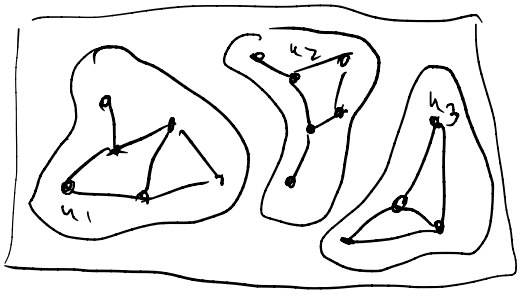
$\Leftrightarrow \nexists$ "back" Kante

DFS auf unger. Graphen

"back" Kanten = "forward" Kanten (umgekehrt durchlaufen)

"cross" Kanten: nicht möglich

Zusammenhangskomponenten:



G

DFS Wald