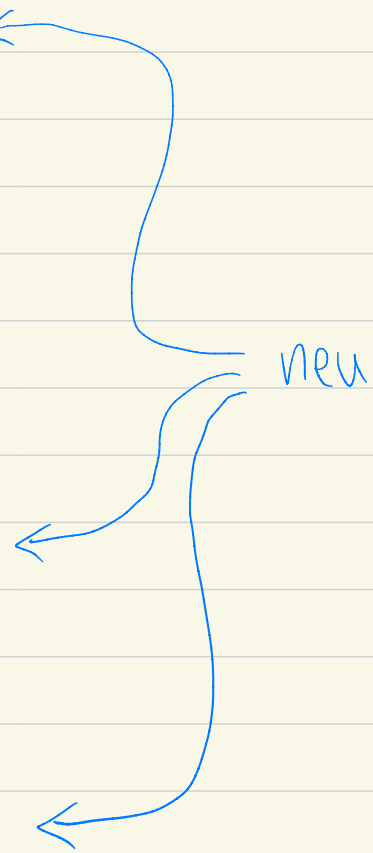
Tiefensuche tiefer verstehen

depth-first search (DFS)

Erinnerung

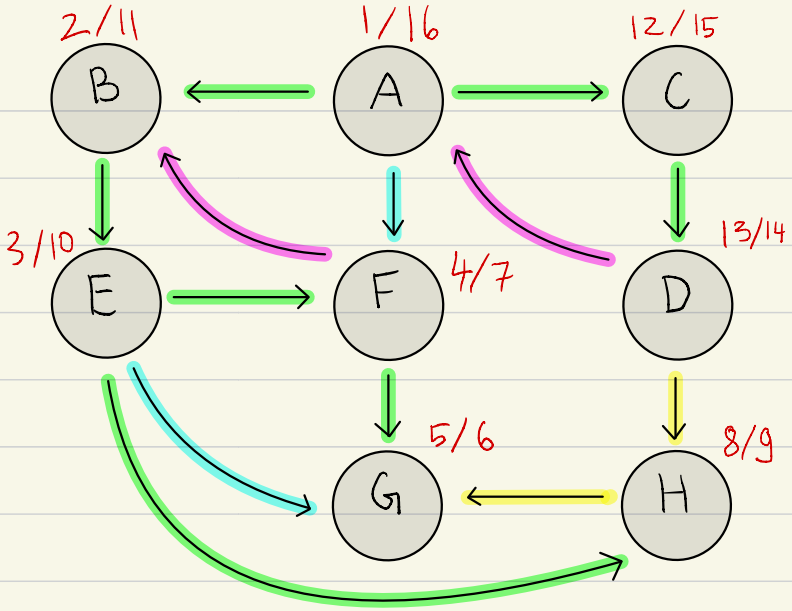
```
Visit(u):
    pre[u] ← T; T ← T+1
    markiere u
    FOR Nachfolger v, unmarkiert
        Visit(u)
    post[u] ← T; T ← T+1

DFS(G):
    T ← 1
    alle Knoten unmarkiert
    FOR u₀ ∈ V, unmarkiert
        Visit(u₀)
```



Neu: merke Start- und Endzeitpunkte aller Visit-Aufrufe

$\rightsquigarrow$ pre / post Zahlen

Beispiel: pre/post



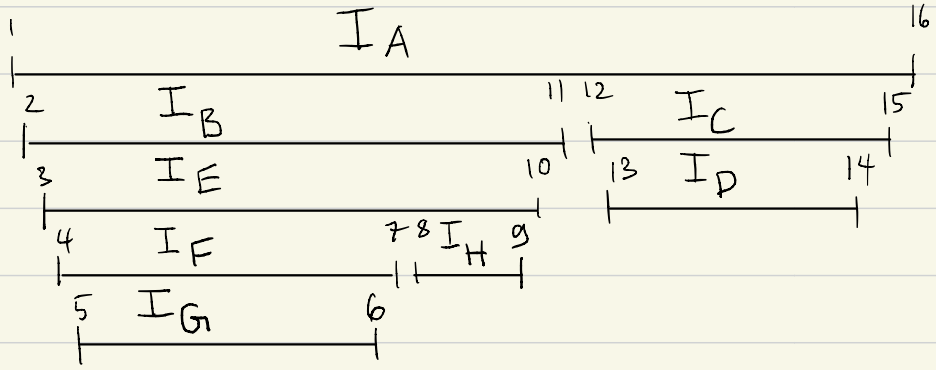
pre-order: ABEFGHCD

post-order: GFHEBDCB

vorherige Vorlesung:

falls G azyklisch, dann ist umgekehrte post-order eine topologische Sortierung

Bildlich: Intervall $I_u = \{ pre[u], \dots, post[u] \}$

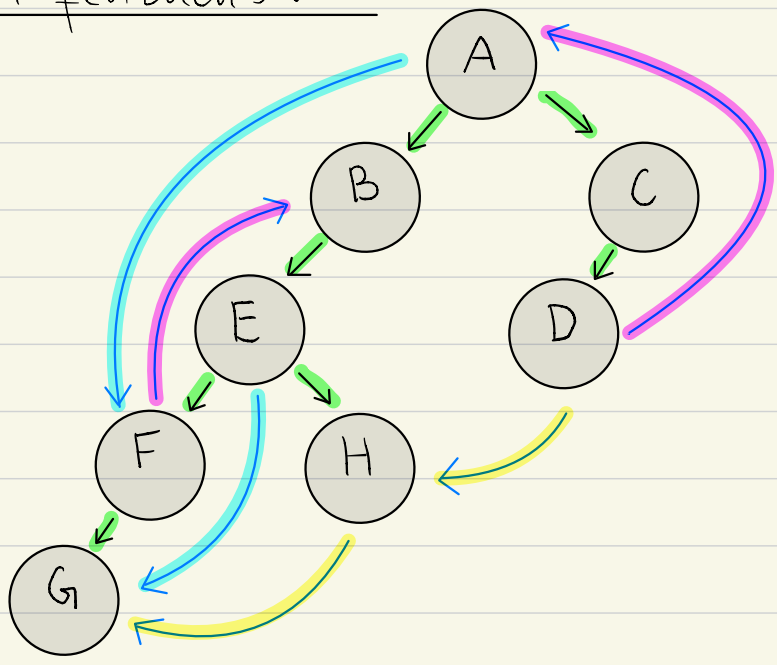


unmassstäblich

Verschachtelung entspricht Rekursionsbaum

Tiefensuchbaum

eigentlich: Tiefensuchwald

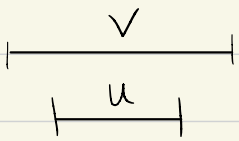


Kanten klassifizieren

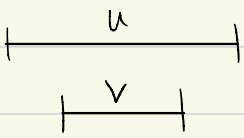
im Tiefensuchbaum: tree

jede andere Kante $(u,v) \in E$: anhand I_u und I_v

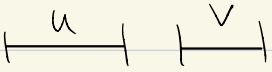
(pre/post Zahlen von u und v)



back z.B.: (F,B), (D,A)

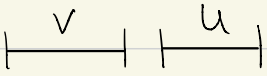


forward (wenn nicht tree) z.B.: (A,F) (E,G)

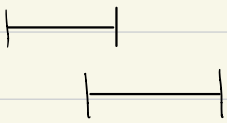


nicht möglich!

Visit(u) würde Visit(v) aufrufen da v unmarkierter Nachfolger von u



cross z.B.: (H,G), (D,H)



nicht möglich!

Aufruf endet erst wenn alle rekursiven Aufrufe beendet

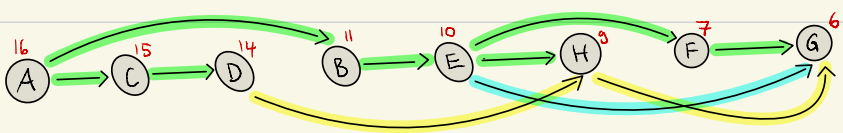
Beobachtungen

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

(2) $\forall$ nicht-back $(u,v) \in E$. post(u) > post(v)

(1),(2) $\rightsquigarrow$ $\nexists$ gerichteter Zyklus $\overset{(1)}{\Rightarrow}$ $\nexists$ back Kante

$\overset{(2)}{\Rightarrow}$ umgekehrte post-order ist topologische Sortierung



Kurzer Korrektheitsbeweis für DFS zum topologischen Sortieren

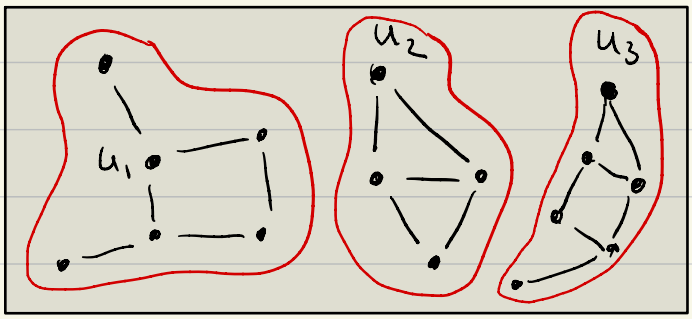
DFS auf ungerichteten Graphen

back = forward Kanten (umgekehrt durchlaufen)

cross Kanten nicht möglich

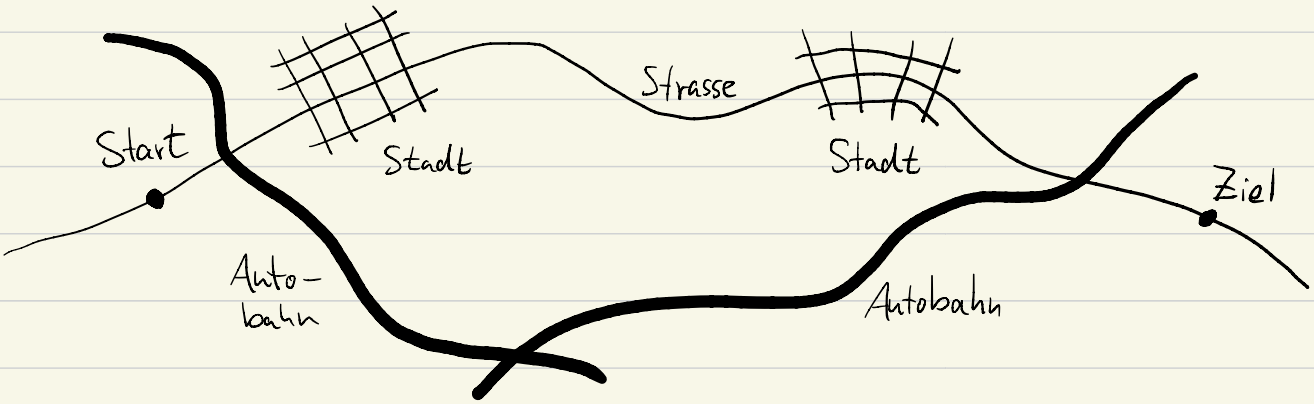
Zusammenhangskomponenten

G



DFS Wald

Strassennetzwerk



gesucht: Route mit möglichst wenig Kreuzungen (auch Auf-/Abfahrten)

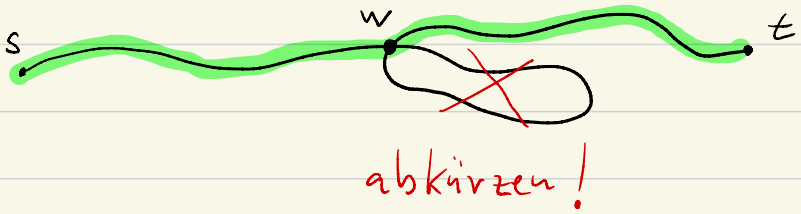
als Graph: Knoten ≃ Kreuzungen

Kanten ≃ Stassenabschnitte ohne Kreuzung

gesucht: Weg im Graph mit wenig Kanten

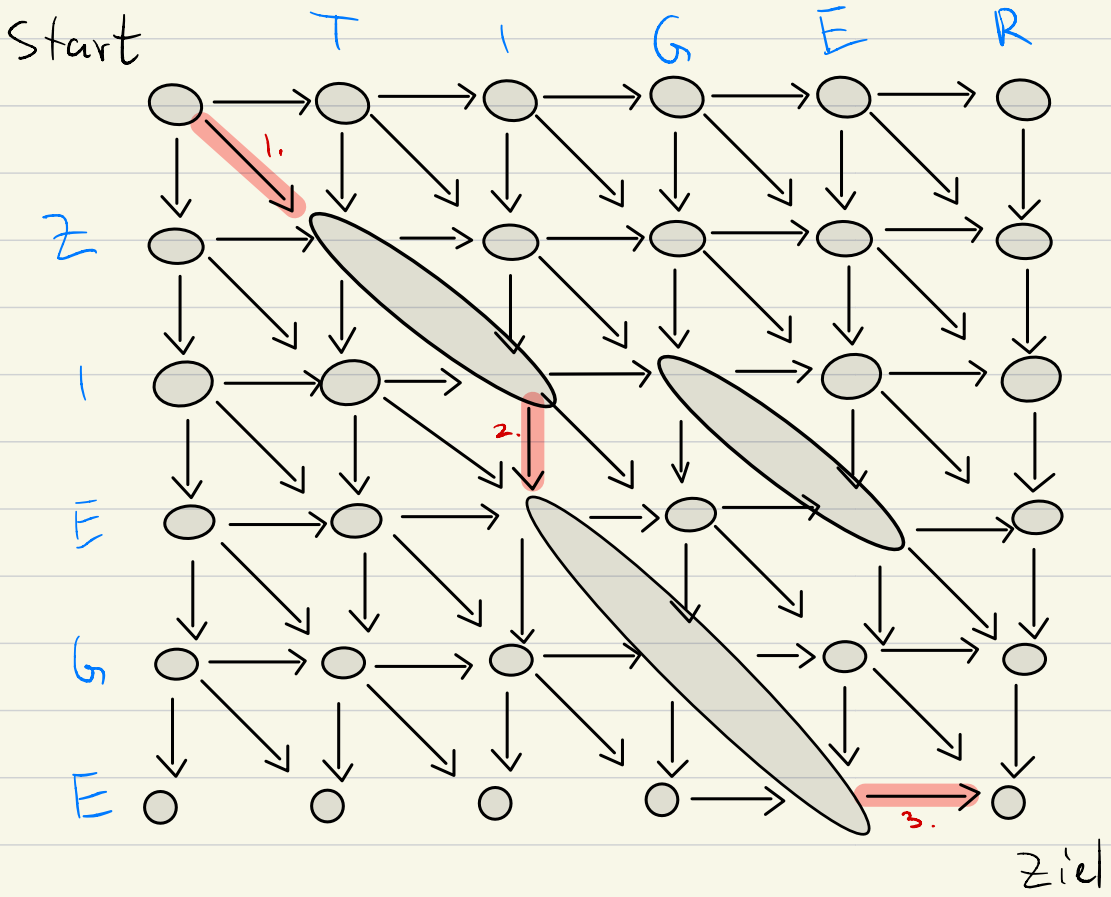
gerichtete Kanten möglich (z.B. Einbahnstrasse)

Beobachtung: Kürzester Weg ist immer Pfad



weiteres Beispiel:

minimale Editierdistanz



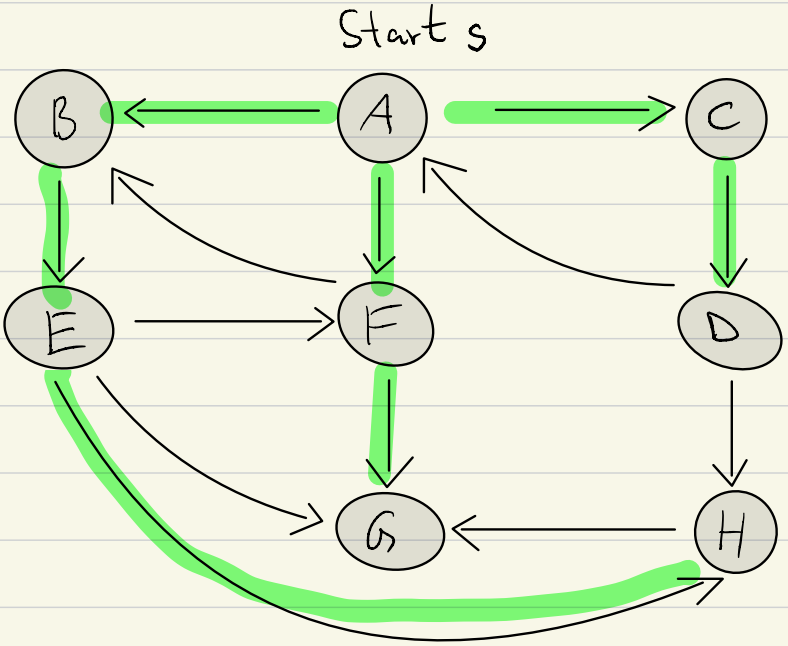
Kanten: mögliche Editieroperationen

kürzester Weg: minimale Sequenz von Editierop.

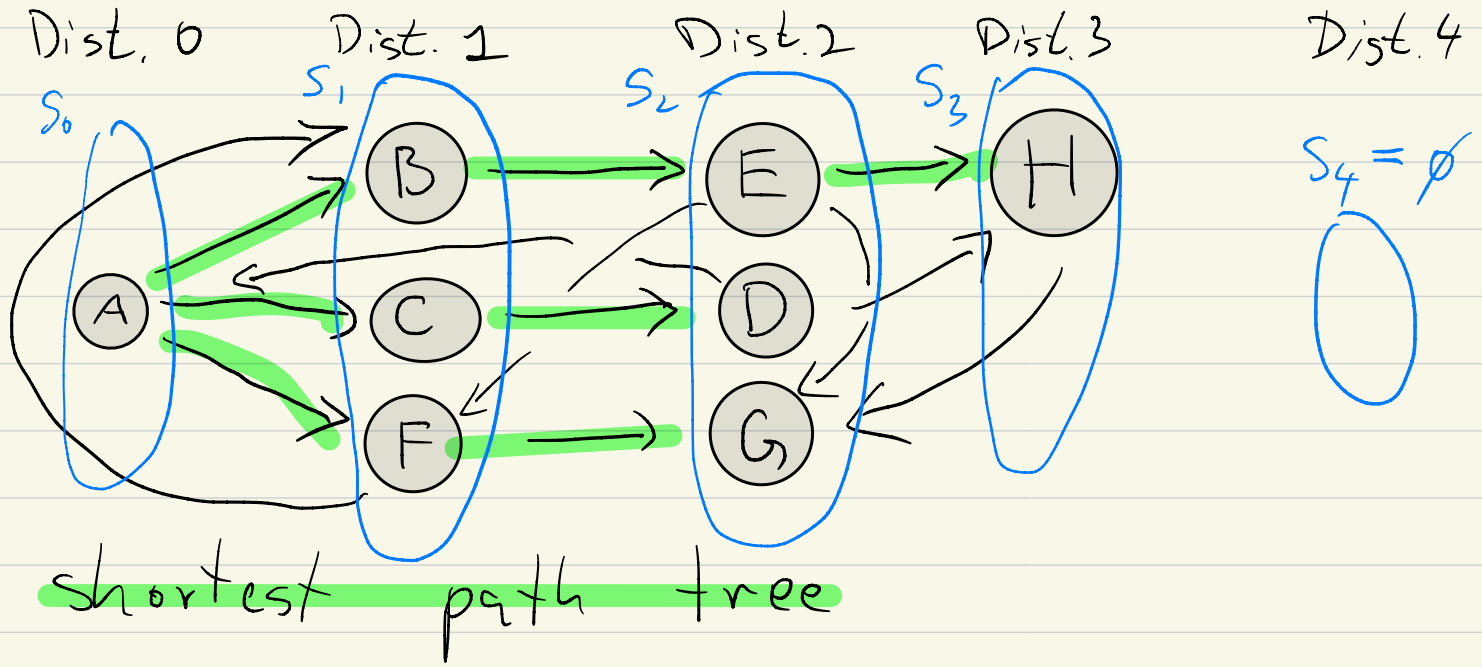
T I G E R
Z I G E R — 1. Tausche T und Z
Z I E G E R — 2. Füge E hinzu
Z I E G E — 3. Entferne R

Definition: ger. Graph $G = (V, E)$, $n = |V|$

Distanz: $dist(u,v)$ = kürzeste Länge eines Wegs von u nach v in G



gesucht: $dist(s,v)$ für alle $v \in V$



level k: $S_k := \{ v \in V \mid dist(s,v) = k \}$

Kante von S_k nach $S_{k'}$ $\Rightarrow$ $k' \leq k+1$

ungerichteter Graph: $k-1 \leq k' \leq k+1$

angenommen: $S_0, \dots, S_{k-1}$ berechnet

wie S_k berechnen?

$v \in S_k \iff v \notin S_0 \cup \dots \cup S_{k-1}$

und v Nachfolger eines Knoten in S_{k-1}

Algorithmus:

$S_0 \leftarrow \{s\}, S_1 \leftarrow \emptyset, \dots, S_{n-1} \leftarrow \emptyset$

n-1 ist grösstmögliche (endliche) Distanz von s

For $k = 1 .. n-1$:

berechne S_k:

For $u \in S_{k-1}$: (2)

For $(u,v) \in E$, $v \notin S_0 \cup \dots \cup S_{k-1}$: (1)

$S_k \leftarrow S_k \cup \{v\}$ (3)

Schnelle Laufzeit

(1) markiere Knoten sobald Distanz bekannt

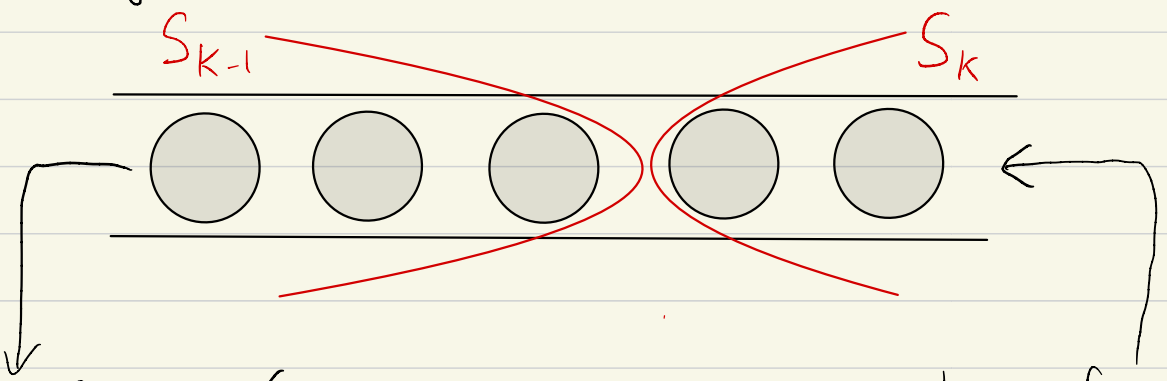
- gemeinsame Datenstruktur um

(2) S_{k-1} abzuarbeiten

(3) S_k aufzubauen

Schlange Q (queue)

im Speicher z.B. als Array mit Zähler



entfernen / dequeue

hinzufügen / enqueue

FIFO: first-in first-out

LIFO: last-in first-out für Stapel

breath-first search / Breitensuche

BFS(s): [Zuse '45, Moore 50er]

$Q \leftarrow \{s\}$ • $dist[s] \leftarrow 0$

$enter[s] \leftarrow 0$; $T \leftarrow 1$

WHILE $Q \neq \emptyset$

$u \leftarrow dequeue(Q)$

$leave[u] \leftarrow T$; $T \leftarrow T+1$

FOR $(u,v) \in E$, $enter[v]$ nicht zugewiesen

$enqueue(Q,v)$ • $dist[v] \leftarrow dist[u]+1$

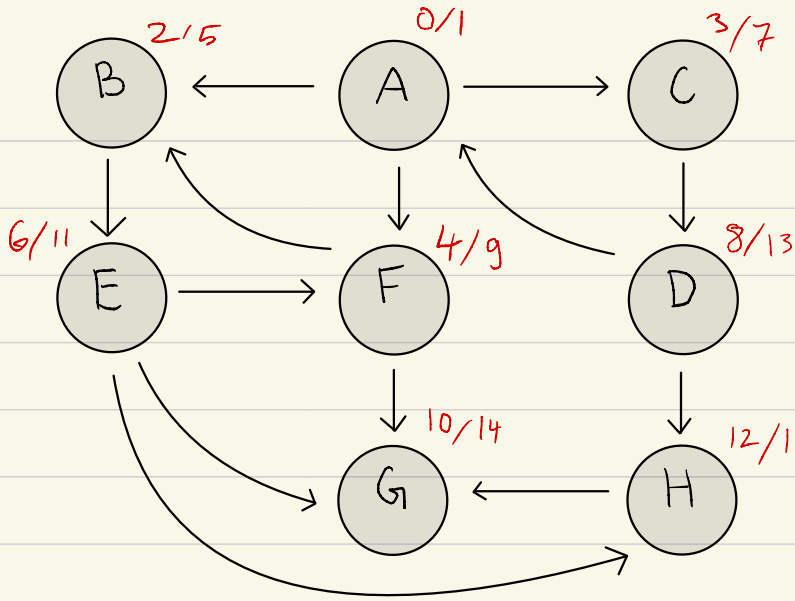
$enter[v] \leftarrow T$; $T \leftarrow T+1$

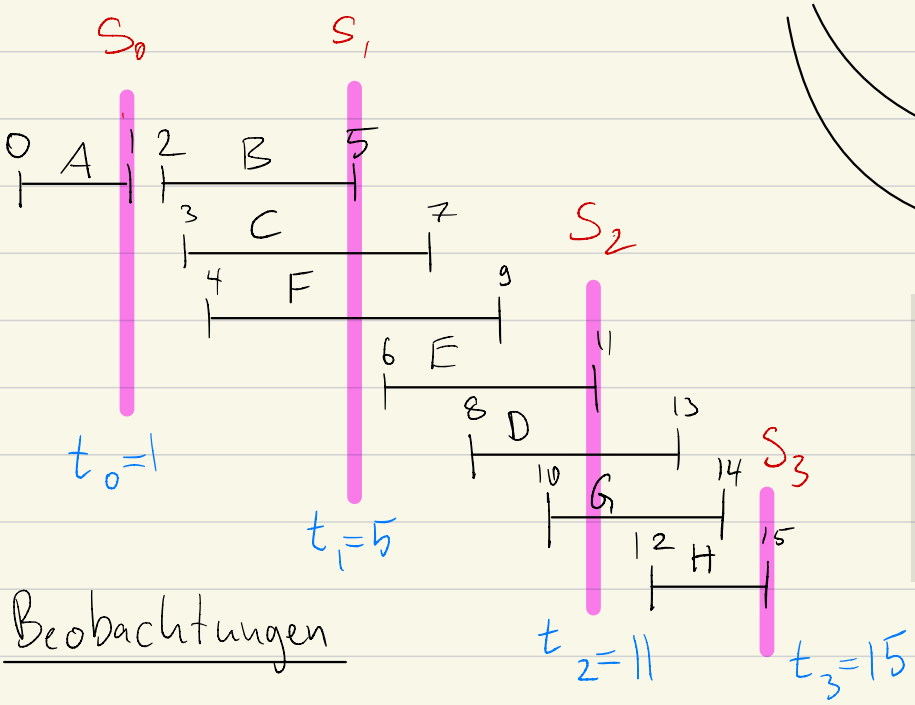
- merke wann Knoten Schlange betreten und verlassen
- Knoten darf Schlange nicht mehrmals betreten

Bemerkung: DFS kann ähnlich iterativ implementiert werden mit Stapel statt Schlange

Beispiel

BFS(A) enter / leave





Intervalle: (vgl. Tiefensuche)

$$I_v := \{ enter[v], \ldots, leave[v] \}$$

Beobachtungen

$enter[v] < leave[v]$

$enter\text{-}order = leave\text{-}order$ (wegen LIFO Eigenschaft von S)

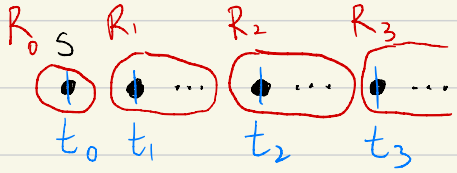
$t_k = \min \{ leave[v] \mid dist(s,v) \geq k \}$ erstes Mal dass ein Knoten mit dist $\geq k$ dequeued wird

$t_0 = 1 \leq t_1 \leq \cdots \leq t_{n-1} \leq t_n = \infty$ ⇝ definieren Epochen für Verlauf von BFS(s)

$R_k := \{ v \mid t_k \leq leave[v] < t_{k+1} \}$ Knoten, die in Epoche k Schlange verlassen

Bildlich:

leave-order



Behauptung $\forall k \in \mathbb{N}_0.$ $\underbrace{S_k = R_k = \{v \mid t_k \in I_v\}}_{\text{Aussage } A(k)}$ (S_k: Inhalt von Q direkt vor Zeit t_k)

Beweis: per Induktion

$A(0)$: (Ind.anfang)

$S_0 = \{s\}$, $leave[s] = 1$, $t_0 = 1$, $\forall v \neq s.\ leave[v] > 1$

$\rightsquigarrow t_1 > 1$, $R_0 = \{s\}$, $\{v \mid t_0 \in I_v\} = \{s\}$ ✓

$A(0), \dots, A(k) \Rightarrow A(k+1)$: (Ind. Schritt)

Q verlassen vor Zeit t_k: $S_0 \cup \dots \cup S_{k-1}$ $(leave[v] < t_k)$

in Q zur Zeit t_k: S_k $(enter[v] \leq t_k \leq leave[v])$

} Ind. hyp.

Zeit t_k bis t_{k+1}:

Q verlassen: S_k $(t_k \leq leave[v] < t_{k+1})$

Q betreten: alle Nachfolger von S_k nicht in $S_0 \cup \dots \cup S_k$

$\rightsquigarrow$ genau S_{k+1} (siehe Charakterisierung auf S. 8)

in Q zur Zeit t_{k+1}: $S_k \setminus S_k \cup S_{k+1} = S_{k+1}$ ✓

Q verlassen von Zeit t_{k+1} bis t_{k+2}: S_{k+1} ✓

} $A(k+1)$

Laufzeitanalyse

$T_u :=$ Laufzeit der WHILE Iteration für Knoten u

$$T_u \leq O(1 + \deg_{out}(u))$$

$$\sum_{u \in V} T_u \leq O\left(\underbrace{\sum_{u \in V} (1 + \deg_{out}(u))}_{} \right)$$

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

$$= |E| \quad \text{(Handschlaglemma)}$$

$$\rightsquigarrow \sum_{u \in V} T_u \leq O(|V| + |E|)$$

$$\rightsquigarrow \text{BFS}(s) \text{ Laufzeit } O(1) + \sum_{u \in V} T_u \leq O(|V| + |E|)$$