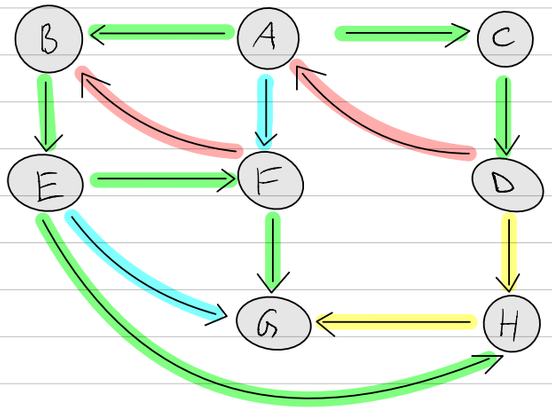
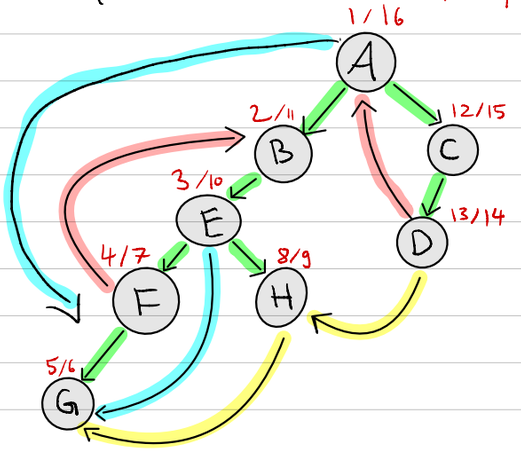
Graph



Tiefensuchbaum (-wald eigentlich)

pre/post



Beobachtungen

∃ "back" Kante ⇒ ∃ ger. Zyklus (1)

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

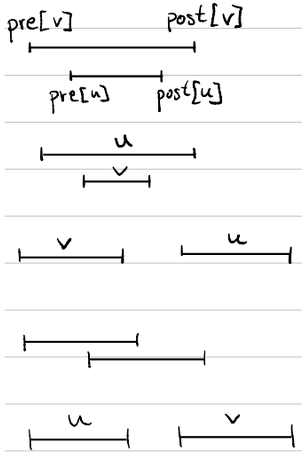
(1),(2) ⇝ ∄ ger. Zyklus ⇔ umgekehrte post-order ist topologische Sortierung

⇔ ∄ "back" Kante

Klassifizierung der Kanten

tree (im Tiefensuchbaum)

andere Kanten $(u,v) \in E$ anhand pre/post Zahlen



back

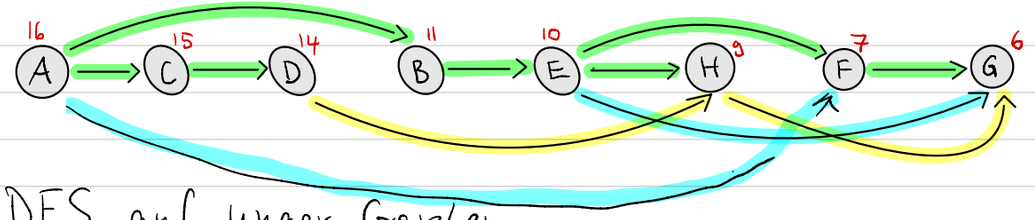
forward (wenn nicht tree)

cross

nicht möglich (rekursive Aufrufe verschachtelt)

nicht möglich (Kante (u,v) wird betrachtet bevor Aufruf für v endet)

depth first search
Tiefensuche

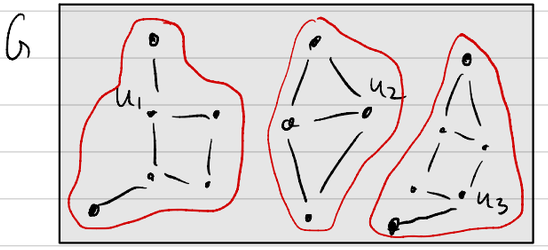


DFS auf unger. Graphen

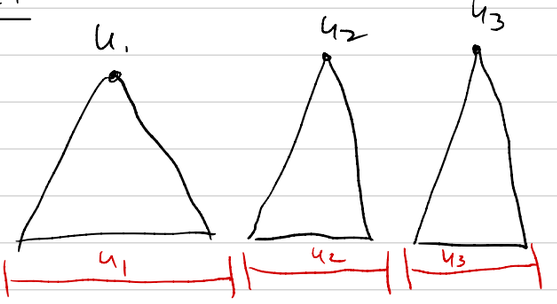
back = forward Kanten (umgekehrt durchlaufen)

cross Kanten nicht möglich

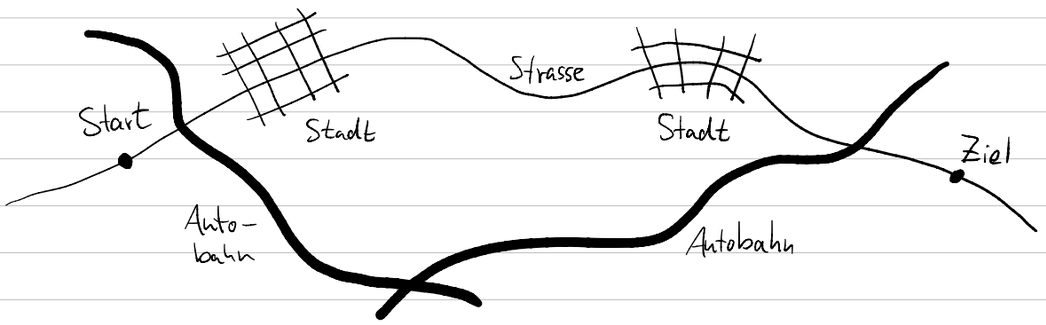
Zusammenhangskomponenten



DFS Wald



Strassennetzwerk



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

als Graph: Knoten ≃ Kreuzungen (und Start/Ziel)

Kanten ≃ Strassenabschnitte ohne Kreuzungen

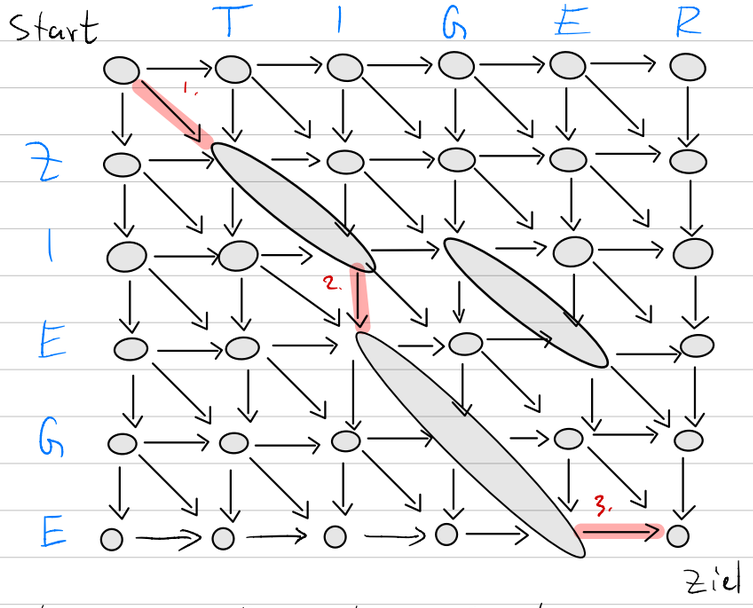
gesucht: Weg im Graph mit wenig Kanten

gerichte Kanten möglich (Einbahnstrasse)

Beobachtung: kürzester Weg ist Pfad

weiteres Beispiel

minimale Editierdistanz



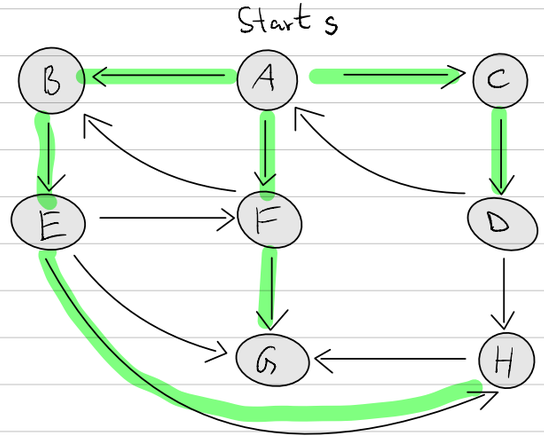
Kanten: mögliche Editieroperation

kürzester Weg: minimale Segmenz v. Editierop.

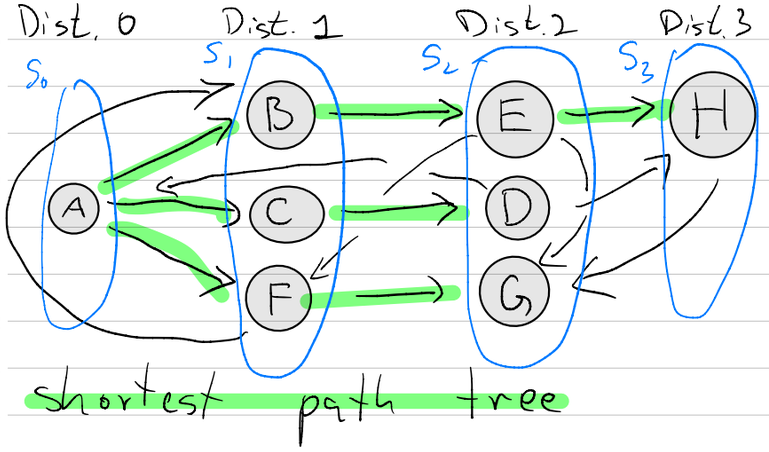
TIGER
ZIGER — 1. Tausche T und Z
ZIEGER — 2. Füge E ein
ZIEGE — 3. Lösche R

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 $S_{k'}$ $\Rightarrow$ $k' \leq k+1$

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

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

wie S_k berechnen?

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

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

Algorithmus:

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

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

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 von Knoten in S_{k-1}

Algorithmus:

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

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)

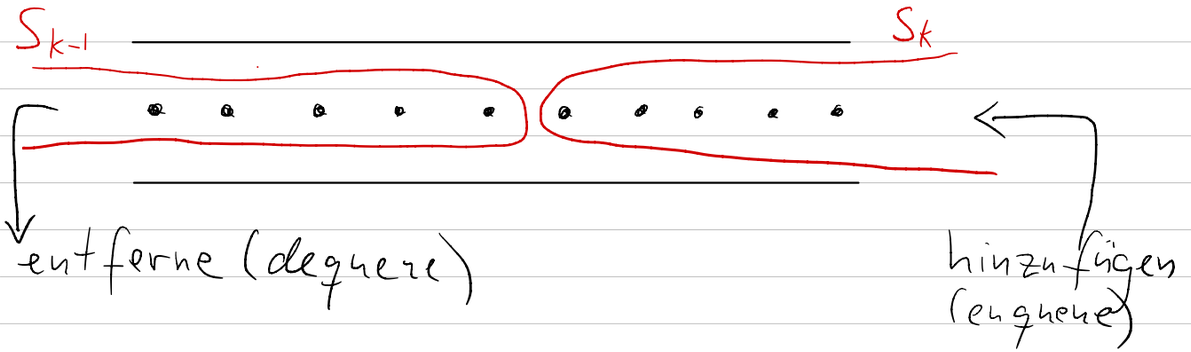
$S_k \leftarrow S_k \cup \{v\}$ (3)

Schnelle Laufzeit

(1) markiere Knoten sobald Distanz bekannt

- gemeinsame Datenstruktur um
  - (2) S_{k-1} abzuarbeiten
  - (3) S_k aufzubauen

Schlange (queue)



BFS(s): (Breitensuche, breadth first search)

$S \leftarrow \{s\}$

while $S \neq \emptyset$

$u \leftarrow$ dequeue(S)

if u nicht markiert

markiere u

For $(u,v) \in E$, v nicht markiert

enqueue(S,v)

[parent$[v] \leftarrow u$ falls parent$[v]$ noch undefiniert]

⚠ Knoten können mehrmals in S auftreten

BFS(s): (Breitensuche, breadth first search)

$S \leftarrow \{s\}$

while $S \neq \emptyset$

Knoten können mehrmals in S auftreten

$u \leftarrow$ dequeue(S)

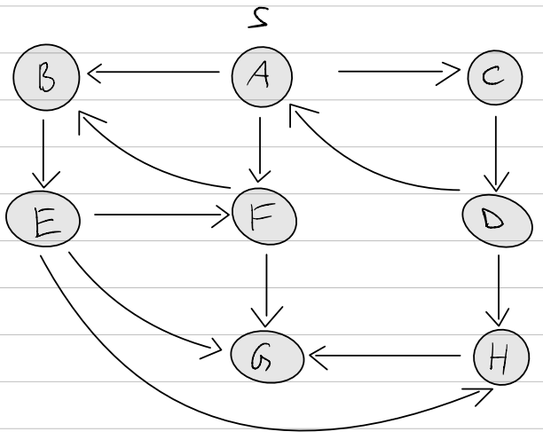
if u nicht markiert

markiere u

For $(u,v) \in E$, v nicht markiert

enqueue(S, v)

[parent[v] $\leftarrow$ u falls parent[v] noch undefiniert]



S: ~~A~~ ~~B~~ ~~C~~ ~~F~~ ~~E~~ ~~D~~ ~~G~~ ~~G~~ ~~H~~ ~~H~~

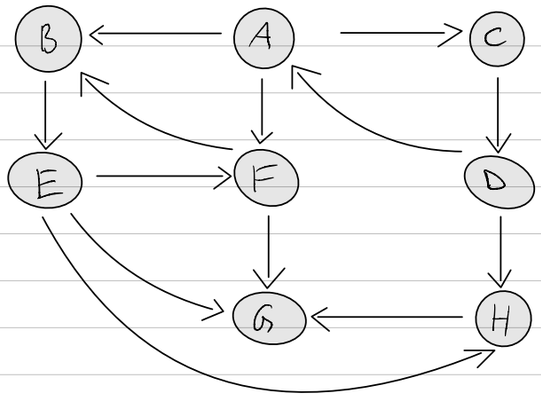
Laufzeit

$O(\#\text{enqueue} + \#\text{dequeue})$

$\#\text{enqueue} \leq |E|$

$\#\text{dequeue} \leq \#\text{enqueue} + 1$

$O(|E| + 1)$ insgesamt


B
A
C
E
F
D
G
H