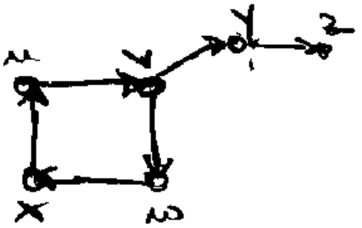
5. Anwendung Tiefensuche: Starke Zusammenhangskomponenten

Starke Zusammenhangskomponenten beziehen sich immer nur auf gerichtete Graphen. Der Begriff der Zusammenhangskomponente im ungewichteten Graph besagt, daß 2 Knoten zu einer solchen Komponente gehören genau dann wenn man zwischen ihnen hin- und hergehen kann. Die analoge Sache führt bei gerichteten Graphen auf starke Zusammenhangskomponenten.

Einige Beispiele:



Starke Zusammenhangskomponente

y kann nicht dabei sein. $y \to z$ ist keine starke Zusammenhangskomponente, also sind y und z alleine. Interessante Beobachtung: Schließlich gehört nicht jede Kante zu einer starken Zusammenhangskomponente.

Die ganze Graph ist eine starke Zusammenhangskomponente.

Zu u, w haben wir z.B. die Wege:

$u \to v \to w$, $w \to v \to u$. Die Kanten der Wege sind alle verschieden, die Knoten nicht!

Damit sind die Vorbereitungen für folgende offizielle Definition getroffen:

<u>Definition</u> (Stark zusammenhängend)

Sei $G = (V, E)$ gerichtet, so ist

G stark zusammenhängend

$\Leftrightarrow$ Für alle $u, v \in V$ gibt es Wege

$u \longrightarrow v$ und $v \longrightarrow u$. $\square$

Also noch einmal: Ist $\overset{v_i}{\circ}\longrightarrow\overset{u}{\circ}$ stark zusammenhängend ?

Es bietet sich an, einen Graphen der nicht stark zusammenhängend ist, in seine stark zusammenhängenden Teile aufzuteilen.

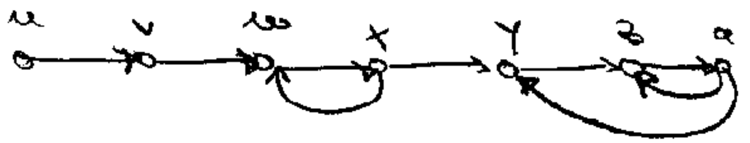
<u>Definition</u> (Starke Zusammenhangs-komponente, starke Komp.)

Ist $\mathcal{G} = (V, E)$ gerichteter Graph.
Ein Teilgraph $H = (W, F)$ von $\mathcal{G}$,
d.h. $W \subseteq V$ und $F \subseteq E$ ist eine
starke Zusammenhangskomponente

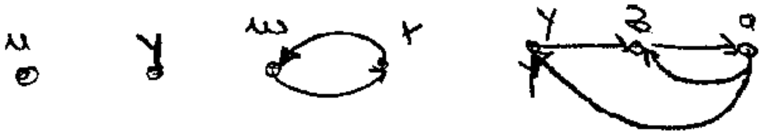
$\Leftrightarrow$

H ist ein maximaler induzierter Teilgraph von $\mathcal{G}$ der stark zshg. ist.
Man sagt auch: Starke Komponente. $\square$

Was soll das maximal? Wir haben induzierten Anordnung auf Teilgraphen von G:
Sei $H = (W, F)$, $H' = (W', F')$, dann H kleiner-gleich H' gdw. H Teilgraph von H', das heißt $W \subseteq W'$. Ein maximaler stark zshg. Teilgraph ist einer, der keinen weiteren stark zshg. über sich hat. Also

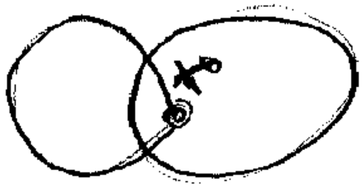


dann sind die starken Komponenten



Nicht  , obwohl stark zshg., wegen der Maximalität.

Man sieht noch einmal: Jeder Knoten gehört zu genau einer starken Komponente. (Warum kann ein Knoten nicht zu 2 verschiedenen starken Komponenten gehören? Etwa so:



)

Wir gehen das Problem an, zu testen, ob ein Graph stark zusammenhängend ist, bzw. die starken Komponenten zu finden. Ein einfacher Algorithmus wäre etwa: Prüfe für je 2 Knoten u, v, ob es einen Weg von u nach v und von v nach u gibt. Etwas genauer:

Eingabe: $\mathcal{G} = (V, E)$, $V = \{1, \dots, n\}$

1. Generiere alle Paare (u,v) von Knoten mit $u < v$.
2. Für jedes (u,v) aus 1. führe aus:

   BFS$(\mathcal{G},u)$, BFS$(\mathcal{G},v)$ ...

   Teste, ob v und u gefunden werden.

   Falls nicht, Ausgabe: "nicht st. zshg. (wg. (u,v))"

3. Ausgabe: "st. zshg".

Laufzeitabschätzung:

1. $O(|V|^2)$
2. $O(|V|^2 \cdot (|V|+|E|))$ also $O(|V|^2 \cdot |E|)$, sofern $|E| \geq \frac{1}{2}|V|$.

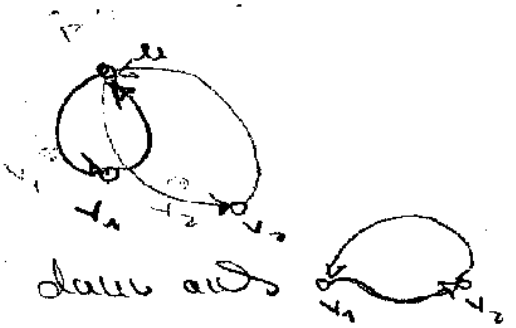
   Kann $O(|V|^4)$ sein.

Es geht etwas besser:

1. BFS(G, u) für einen Knoten u, so modifiziert, daß alle gefundenen v auf Liste gespeichert.

2. Für jedes in 1. gefundene v führe aus:

BFS(G, v). Teste ob u neu gefunden wird.



Zeitabschätzung:

1. $O(|V| + |E|)$

2. $O(|V| \cdot (|V| + |E|))$ also $O(|V| \cdot |E|)$

sofern $|E| \geq \frac{1}{2} |V| \cdot |V| + |E|$ kann bis $|V|^3$ gehen ($|V| = 10$, dann $|V|^3 = 1000$!)

Bemerkung

Sei $G = (V, E)$ gerichteter Graph und sei $v \in V$. G ist stark zusammenhängend gdw. für alle $w \in V$ gilt $v \rightsquigarrow w$, $w \rightsquigarrow v$.

Beweis:

„$\Rightarrow$" klar nach Definition

„$\Leftarrow$" Sind $x, y \in V$, dann

$$x \rightsquigarrow v \rightsquigarrow y \quad \text{und} \quad y \rightsquigarrow v \rightsquigarrow x$$

nach Voraussetzung. $\square$

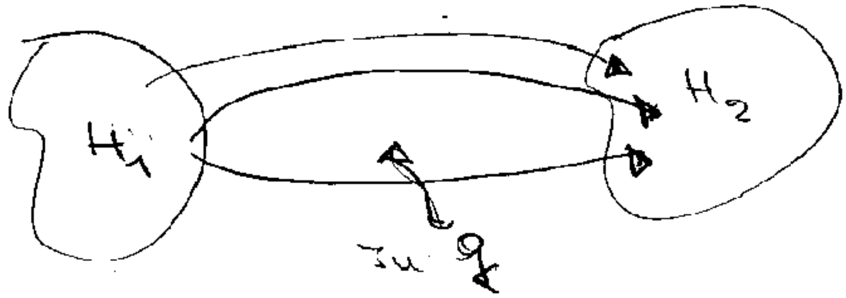
Finden von starken Komponenten in $O(|V| + |E|)$!

Schlüsselbeobachtung ist: Sind

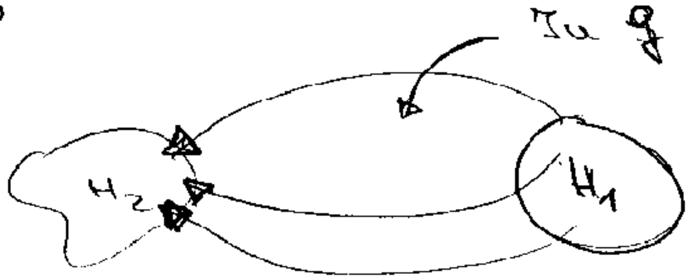
$H_1 = (W_1, F_1)$, $H_2 = (W_2, F_2)$

zwei starke Komponenten von

$G = (V, E)$, dann



oder



und allgemeiner:

Satz

Fassen wir die starken Komponenten als einzelne Knoten auf, und verbinden sie gdw. sie in G verbunden sind, so ist der entstehende gerichtete Graph kreisfrei.

Beweis

Ist $(H_1, H_2, \ldots, H_k, H_1)$ ein Kreis auf der Ebene der Komponenten, so gehören alle H_i zu einer Komponente. $\square$

Können Komponenten topologisch sortieren!

Immer ist nach Tiefensuche: (5.12)

größte Beendezeiten

kleinste Beendezeiten

Alles außer Rückwärtskante:

Ordnen wir die Knoten einmal nach absteigender Beendezeit, dann immer:

$$v_1 \overbrace{\qquad\qquad}^{\text{von } H_1} v_2 \overbrace{\qquad\qquad}^{H_2 \text{ oder } H_1} v_3 \overbrace{\qquad\qquad}^{H_3, H_2, H_1}$$

$f[v_1]$ am größten, $v_1 \in H_1$.

$f[v_2]$ am größten in H_2

$f[v_3]$ am größten in H_3

v_1 = Knoten mit kleinster Anfangszeit in H_1

v_2 = Knoten mit kleinster
Anfangszeit in H_2

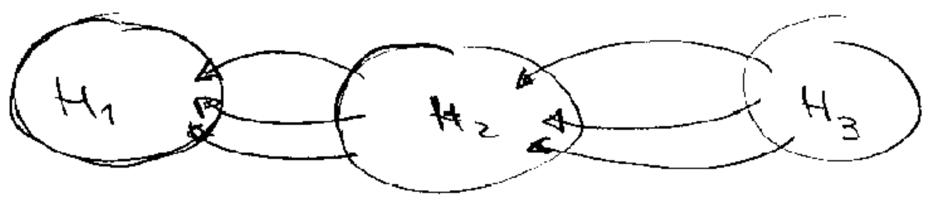
v_3 = Knoten mit kleinster
Anfangszeit in H_3.

(W.-W.-Satz) Bem.: Max. Beendzeit in Komp. läßt top. Sortierung erkennen.

Wie kann man aber diese
v_1, v_2, v_3 erkennen?

Fingen wir in H_3 an, dann
H_2, dann H_1, dann v_3, v_2, v_1
Wurzeln der Bäume. Aber
nicht, wenn H_1, H_2, H_3.
Reinlaufen tritt auf.

Deshalb: Eine weitere Tiefensuche mit Umkehrgraph:



Hauptschleife von DFS(G) in obigen Reihenfolge:

$v_1 \overbrace{\qquad}^{H_1} v_2 \overbrace{\qquad}^{H_1, H_2} v_3 \overbrace{\qquad}^{H_1, H_2, H_3}$.

Komponenten kommen nach topologischer Sortierung.

Dann

DFS-v(v_1) : genau H_1

v_2 steht vorne auf der Liste.

DFS-v(v_2) : genau H_2

v_3 vorne auf der Liste

DFS-v(v_3) : genau H_3.

Algorithmus (starke Komponenten)

St.-Komp(G)

Eingabe $G=(V,E)$ gerichteter Graph.

1. DFS(G). Hat Liste nach absteigender Beendezeit

$V=(v_1, v_2, \dots, v_m)$, $f[v_1] > f[v_2] \dots > f[v_m]$.

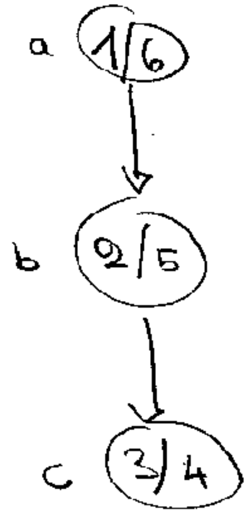
Zuletzt beendet. Zuerst beendet

2. Drehe Kanten in G um. Graph G^u.

3. DFS(G^u) mit Hauptschleife nach Liste V. Jedes DFS-v(v) in Hauptschleife ergibt starke Komponente. Zeit: $O(|V|+|E|)$! □

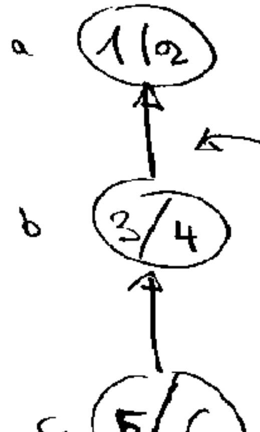
Beispiel

1. Tiefensuche



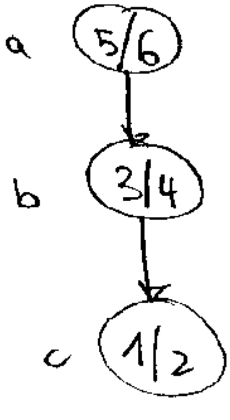
$V = (a, b, c)$

2. Tiefensuche



Kein Rauslaufen

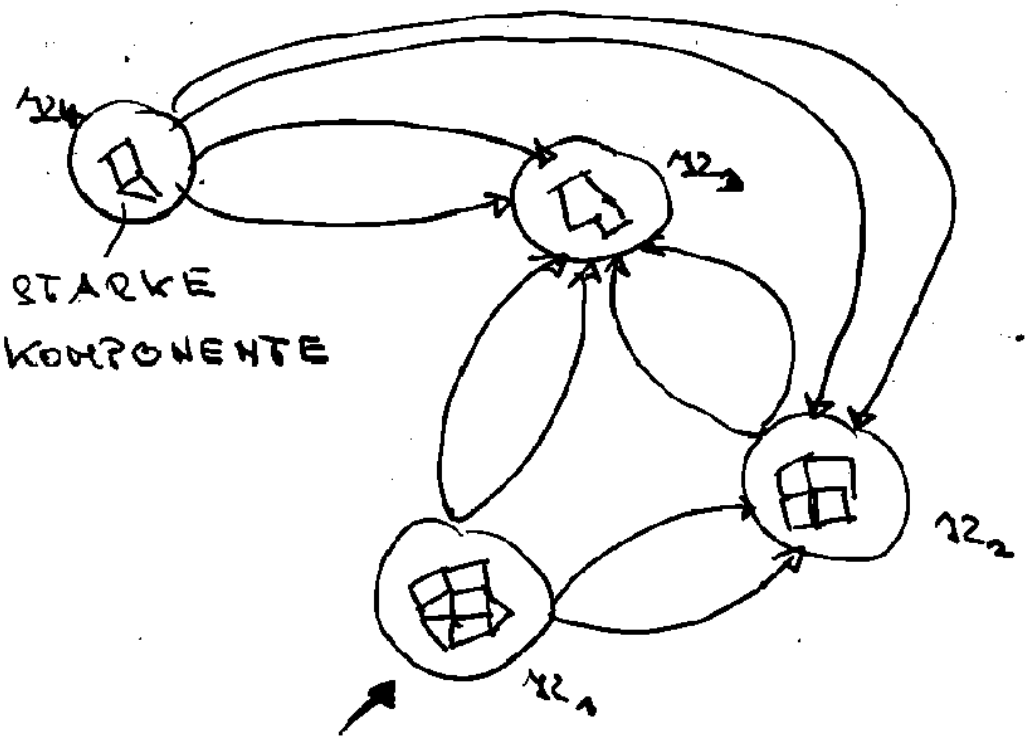
1. Tiefensuche



2. Tiefensuche wie vorher.

Diese Seite mußte aus rechtlichen Gründen entfernt werden!

MOTIVATION DES ALGORITHMUS



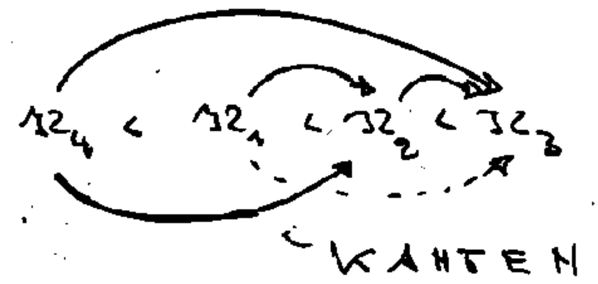
ERSTE TIEFENSUCHE
FÄNGT HIER AN.

- EINE TIEFENSUCHE ENTDECKT IN JEDEM FALL DIE KOMPONENT EINES KNOTENS
- TIEFENSUCHE KANN ABER RAUSLAUFEN!
- TIEFENSUCHE LÄSST TOPOLOGISCHE SORTIERUNG AUF DEN KOMPONENTEN ERKENNEN: NACH MAX. BEENDEZEIT IN KOMPONENTE:

$$K_4 < K_1 < K_2 < K_3$$

MAX. BEENDEZEIT

EGAL OB ZUERST NACH K_2 ODER K_3.

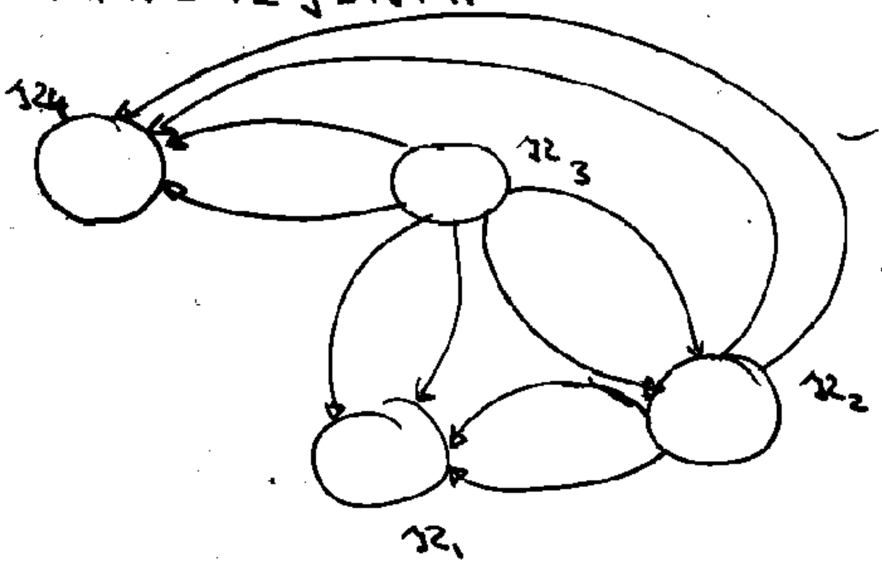
- 

- BEI BEGINN IN SZ_1 BEKOMMEN WIR:

  $SZ_1 < SZ_4 < SZ_2 < SZ_3$.

- WIE KANN MAN DAS RAUSLAUFEN JETZT VERHINDERN?

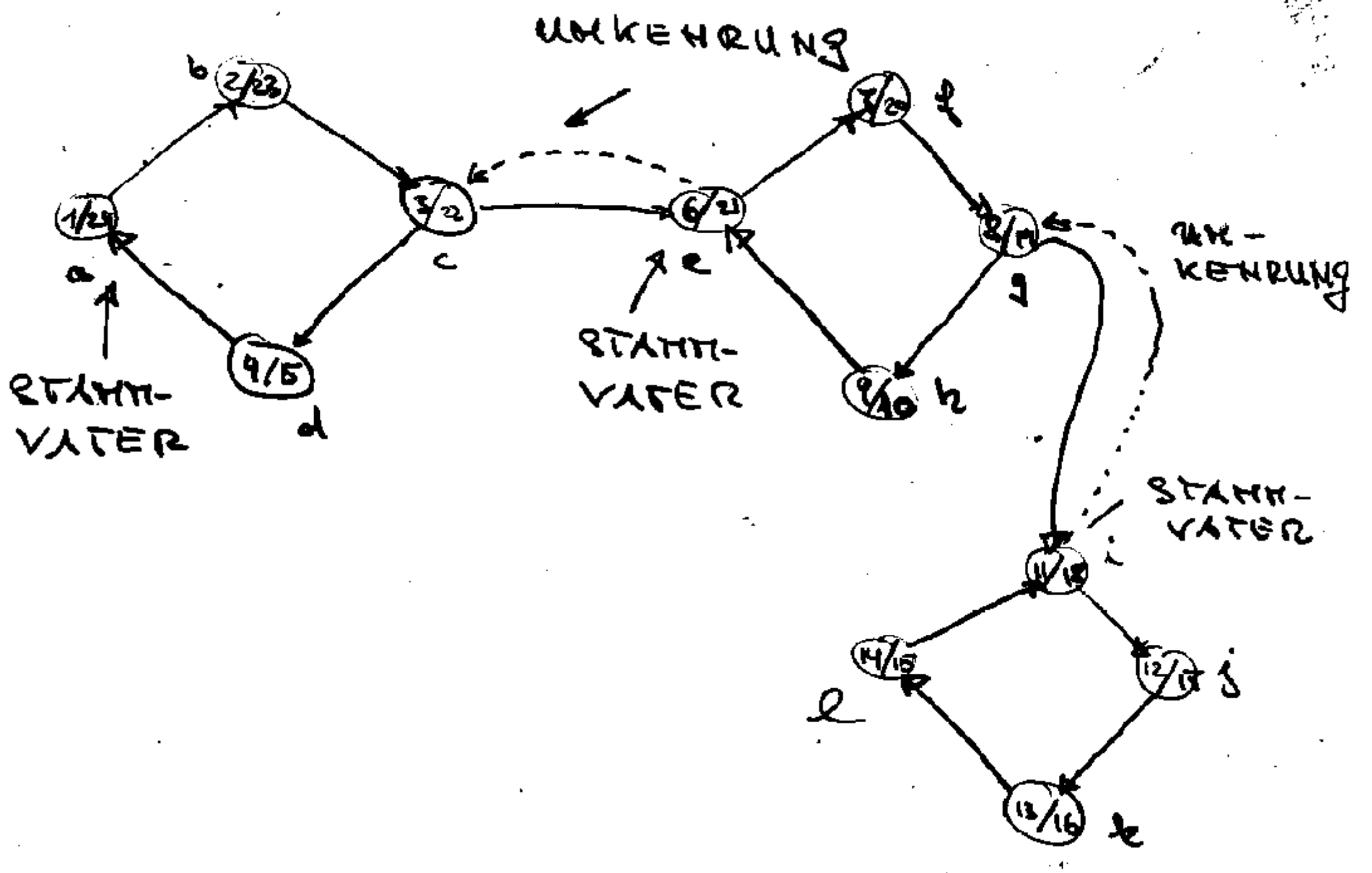
- UMKEHRGRAPH



KEIN RAUSLAUFEN, DA SZ_4 SCHWARZ VOR SZ_2.

- HAUPTSCHLEIFE GEMÄSS TOPOLOGISCHER SORTIERUNG, D. H. ABFALLENDE BEENDEZEIT:

  SZ_4, SZ_1, SZ_2, SZ_3



1. TIEFENSUCHE VON STARKE-KOMP(G)

KNOTENLISTE:

(a, b, c, e, f, g, i, j, k, l, h, d)

2. K.

1. KOMPONENTE

STAMMVATER = DER KNOTEN, ÜBER DEN EINE STARKE KOMPONENTE DAS ERSTE MAL BETRETEN WIRD.

ERSTES MAL = KLEINSTE ANFANGSZEIT.

Einige Überlegungen zur Korrektheit. Welcher Knoten einer Komponente wird zuerst betreten?

<u>Definition</u> (Stammvater)

Nach dem Lauf von DFS(G) ist sei für $u \in V$

$\Phi(u)$ = der Knoten, der unter allen von u erreichbaren Knoten die maximale Beendezeit hat.

$\Phi(u)$ heißt Stammvater von u. □

Satz

(a) Es ist col[$\Phi(u)$] = grau bei $d[u]$.

(b) Im G: $\overset{\Phi(u)}{\bullet} \longrightarrow \overset{u}{\circ}$, $\overset{u}{\bullet} \longrightarrow \overset{\Phi(u)}{\circ}$.

Beweis

(a) Klar bei $u = \Phi(u)$.

Sei also $u \neq \Phi(u)$. Wir schließen die anderen Fälle aus.

1. Fall col[$\Phi(u)$] schwarz bei $d[u]$.

Dann $f[\Phi(u)] < d[u] < f[u]$,

im Widerspruch zur Definition, da ja $u \leadsto u$.

5.19

<u>2. Fall</u> col[$\Phi(u)$] weiß bei d[u].

Gibt es zu d[u] weißen Weg

von $\overset{u}{\bullet} \rightsquigarrow \overset{\Phi(u)}{\circ}$, dann

$$d[u] < d[\Phi(u)] < f[\Phi(u)] < f[u]$$

im Wid. zu Definition von $\Phi(u)$.

Gibt es zu d[u] keinen

weißen Weg von $\overset{u}{\circ} \rightsquigarrow \overset{\Phi(u)}{\bullet}$

dann, da ja $\overset{u}{\circ} \rightsquigarrow \overset{\Phi(u)}{\circ}$ sieht

es so aus: col[$\Phi(u)$] = weiß

$$\overset{u}{\circ} \to \overset{u_1}{\bullet} \to \overset{u_2}{\circ} \to \dots \underbrace{\to \overset{u_k}{\circ} \to \overset{\Phi(u)}{\circ}}_{\text{w-Weg.}}$$

col[u_k] = grau bei d[u]

col[u_k] = sch geht nicht.

Es ist v_ℓ der letzte Knoten auf dem Weg mit col[v_ℓ] = grau.

Dann aber wegen w-Weg

$$d[v_\ell] < d[\Phi(u)] < f[\Phi(u)] < f[v_\ell]$$

im Widerspruch zur Def von $\Phi(u)$, da

$$u \longrightarrow u_\ell$$

(b) Mit (a) ist

$$d[\Phi(u)] < d[u] < f[u] < f[\Phi(u)] \quad \square$$

Folgerung

(a) u, v in einer st. Komp. Stammväter

$\Leftrightarrow \Phi(u) = \Phi(v)$ $\Leftrightarrow$ st. Komp's.

(b) $\Phi(u)$ = der Knoten mit kleinster Anfangszeit in der Komp. von u.

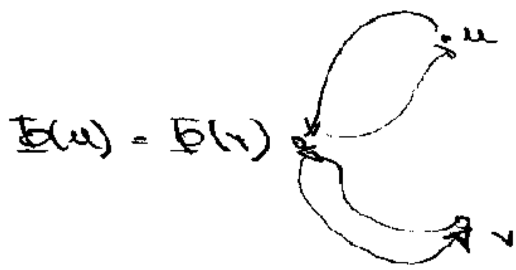
Beweis:

(a) mit letztem Satz und Definition Stammvater.

$$
\begin{tikzcd}
u \arrow[r] \arrow[d, bend right] & \Phi(u) \\
v \arrow[r] \arrow[u, bend right] & \Phi(v)
\end{tikzcd}
$$

Also $\Phi(v) = \Phi(u)$.

Andererseits



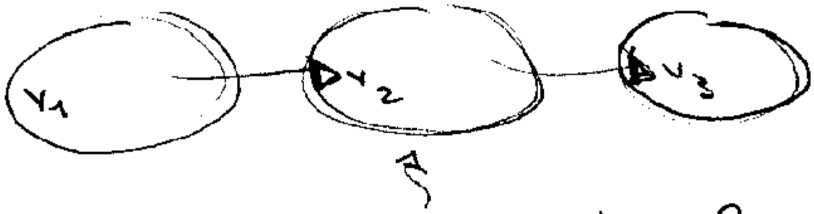
Also u, v in einer Komp.

(b) Sei v mit kleinster Anfangszeit in d. Komp. Dann

$$d[v] < d[u] < f[u] < f[v]$$

für alle u in der Komponente. Alle von u erreichbaren werden vor f[u] entdeckt also vor f[v] beendet. Also $v = \Phi(u)$ □

Beachte noch einmal



1. Tiefensuche fängt z.B. hier an.

$V = (v_1, \dots, v_2, \dots, v_3)$ wg. Beendezeiten.

Die Korrektheit des Algorithmus ergibt sich jetzt aus:

<u>Satz</u>

In der Liste $V = (v_1, \dots, v_n)$ gibt die Position der Stammväter eine topologische Sortierung der st. Komponenten an.

(5.24)

<u>Beweis</u>

Sei also die Liste

$V = (v_1, \ldots, v_n)$

v_i Stammvater

v_k Stammvater

v_l Stammvater

Sei $k < l$, dann gibt es keinen Weg in G: $v_l \longrightarrow v_k$,

denn sonst wäre v_l kein Stammvater, da $f[v_l] > f[v_k]$.

$\square$

5.25

Im Umkehrgraph: gleiche Komponenten, wie vorher. Aber Kanten zwischen Komponenten gegen topologische Sortierung. Also gibt die

2. Tiefensuche die Komponenten aus.