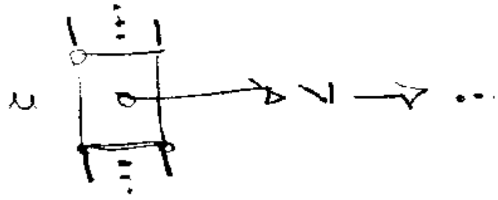
4. Tiefensuche in gerichteten Graphen

Zunächst betrachten wir das Beispiel auf S. 4.2.

Fangen bei u zum Zeitpunkt 1 an. Gehen 1 Kante, entdecken v zum Zeitpunkt 2, (b). Adjazenzliste

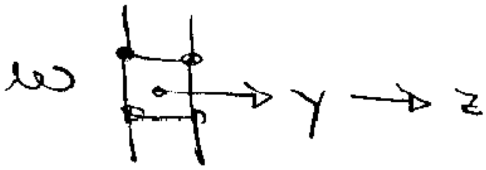


v ist das erste Knoten. Dann wird y in (c) entdeckt. u, v, y sind offen = grau. In (d) entdecken wir die x. Kante $x \longrightarrow y$ wird zwar gegangen, aber y nicht darüber entdeckt.

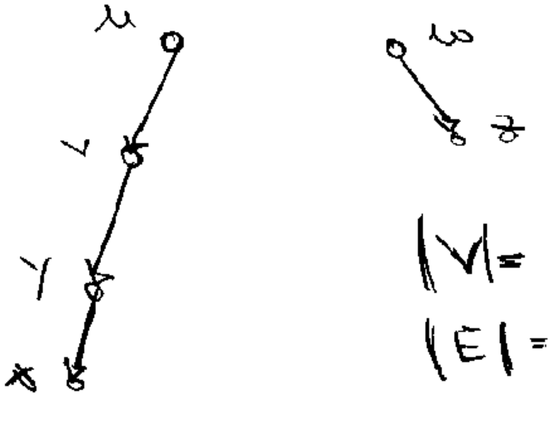
Diese Seite mußte aus rechtlichen Gründen entfernt werden!

4.8

Dann bei w weiter. Adjazenz-liste ist



Tiefensuchwald = Kanten, über die entdeckt wurde.



$|V| = 6$

$|E| = 4 = |V| - 2.$

Wald = Menge von Bäumen

$\pi[x] = y$, $\pi[y] = y$ $\pi[v] = u$, $\pi[u] = u$, (alternativ nil)

$\pi[w] = w$ (alternativ nil)

Algorithmus (Tiefensuche)

Eingabe $G = (V, E)$ in Adj-Listen

Dstg., gerichteter Graph, $n = |V|$.

$d[1 \ldots n]$ = Entdeckzeit (discovery Zeit)

Nicht(!) Entfernung wie vorher.

Knoten wird grau.

$f[1 \ldots n]$ = Beendezeit (finishing)

Knoten wird schwarz.

$\pi[1 \ldots n]$ = Tiefensuchwald, wobei

$$\pi[u] = v \iff \begin{array}{c} v \\ \downarrow \\ u \end{array}$$

$\pi[u]$ = Knoten über den u entdeckt wurde

$col[1 \ldots n]$ = aktuelle Farbe

(4.5)

DFS(G)

1. Für alle $u \in V$ {
 col[u] := weiß;
 π[u] := nil
 }

time := 0

for each $u \in V$ { // Hauptschleife
 if col[u] = weiß {
 DFS-visit(u)
 }

// DFS-v(u) immer
// nur, wenn
// col[u] = weiß.

(4.6)

DFS-visit(u) // Hier ist col[u] = weiß.

1. col[u] := grau // Damit ist u
// entdeckt

2. d[u] = time ; time = time + 1

3. for each v ∈ Adj[u] { // u wird bearbeitet

4. if col[v] = weiß { // (u,v) unbearbeitet

5. π[v] = u // v entdeckt

6. DFS-visit(v)

}

}

// Bearbeitung von u
// ist hier zu Ende
7. col[u] = schwarz // sind alle Knoten
// auf Adj[u] grau
8. f[u] := time; // oder schwarz, so wird
time := time + 1 // u direkt schwarz.

// Zeitzähler geht hoch
// bei Entdecken und
// beenden.

□

- Es ist

$$\{d[1],\dots,d[m],\ f[1],\dots,f[m]\} = \{1,\dots,2m\}.$$

Man kann über die relativen Reihenfolge nicht viel sagen.

Möglich ist

Einmal:

$d[u]=1$, $d[v]=2$, $d[w]=3$

und

$f[w]=4$, $f[v]=5$, $f[u]=6$.

Aber auch

$d[w]=1$, $f[w]=2$,

$d[v]=3$, $f[v]=4$

$d[u]=5$, $f[u]=6$.

- DFS-visit(u) wird nur dann aufgerufen, wenn col[u] = weiß. Rekursion geht nur in weiße Knoten.

- Nachdem für alle von u aus über weiße(!) Knoten erreichbare Knoten besucht sind, wird $col[u] = schwarz$.

- Im nachhinein war während eines Laufs

  $col[u] = weiß$ solang $time < d[u]$

  $col[u] = grau$ solang $d[u] < time < f[u]$

  $col[u] = schwarz$ solang $f[u] < time$.

Unser Eingangsbeispiel führt zu folgendem Prozeduraufrufbaum:

$$
\begin{tikzcd}
 & \mathrm{DFS}(G) \arrow[dl, no head] \arrow[dr, no head] & \\
\text{DFS-v}(u) \arrow[d, no head] & & \text{DFS-v}(w) \arrow[d, no head] \\
\text{DFS-v}(v) \arrow[d, no head] & & \text{DFS-v}(z) \\
\text{DFS-v}(y) \arrow[d, no head] & & \\
\text{DFS-v}(x) & &
\end{tikzcd}
$$

Sequentielle Bearbeitung.

kein DFS-v(u)

Beachte den Tiefensuchwald:

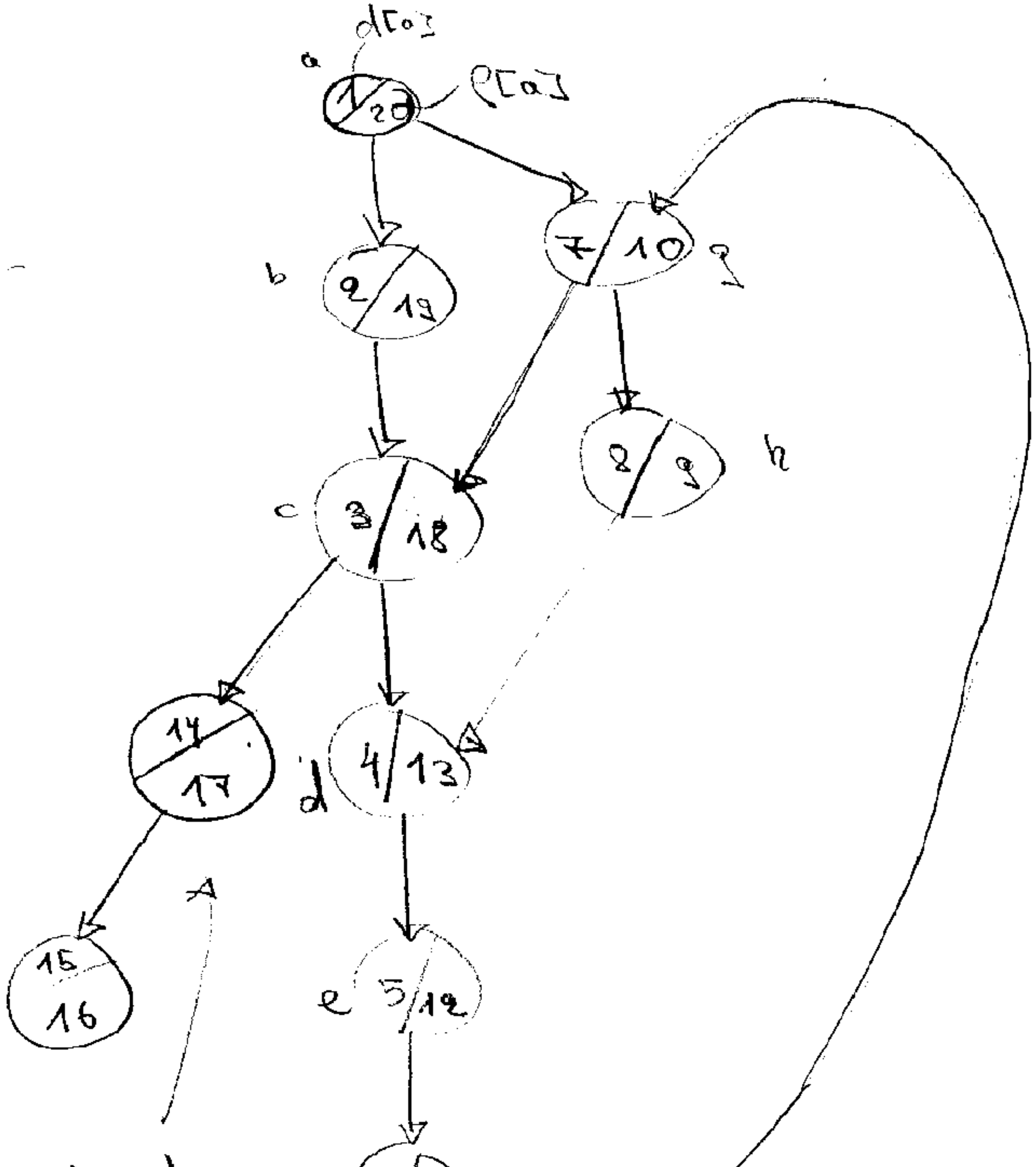
$$
\begin{tikzcd}
u \arrow[d] & w \arrow[d] \\
v \arrow[d] & z \\
y \arrow[d] & \\
x &
\end{tikzcd}
$$

Erzeugung: Präorder (Vater vor Sohn)

Beendigung: Postorder (Sohn vor Vater)

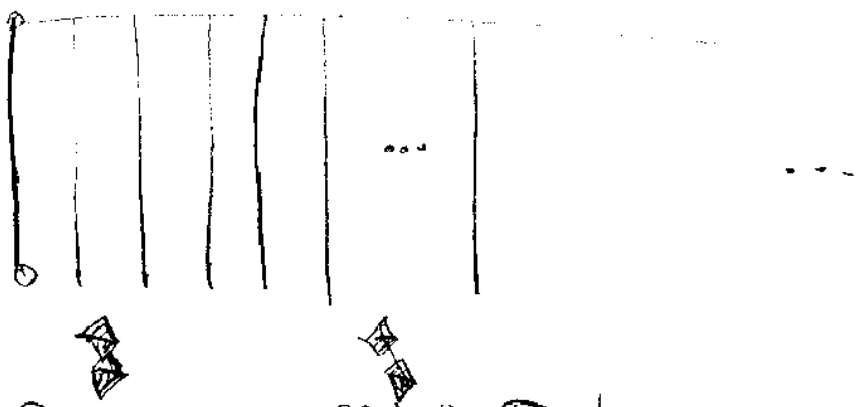
4.10

Betrachten wir noch einmal folgendes Beispiel



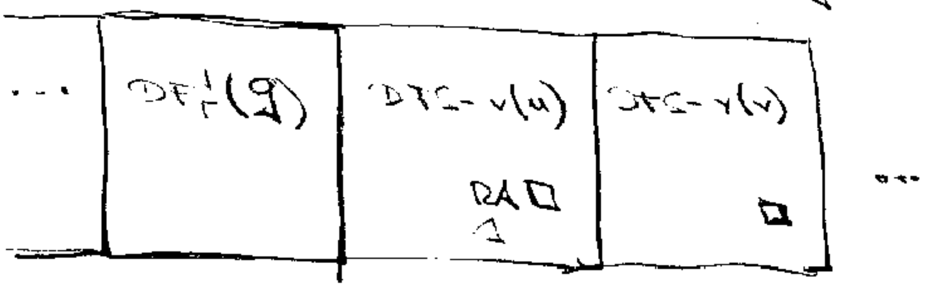
wird über c entdeckt,
nicht über g, obwohl von g erreichbar.

Wie Ausführung des Prozeduraufrufs auf unserem Maschinenmodell der RAM? Organisation des Hauptspeichers als Keller von Frames:



Programm

globale Daten eines Programms, arrays usf. (heap, nicht die Datenobjekte gleichen Namens)



konstante Größe (nur programmabhängig)

Rücksprungadresse

Vgl. AuP Kapitel ?

Verwaltungsaufwand zum
Einrichten und Löschen eines Frames:

O(1), programmabhängig. Hier werden
im wesentlichen Adressen umgesetzt.

Merkregel zur Zeitermittlung:

Durchlauf jeder Programmzeile
inklusive Prozeduraufruf ist O(1)

Bei Prozeduraufruf zählen wir so:

Zeit für den Aufruf selbst
(Verwaltungsaufwand) O(1)

+

Zeit bei der eigentlichen
Ausführung.

Satz

Für $G = (V, E)$ mit $n = |V|$ und $m = |E|$ hat braucht DFS(G) eine Zeit von $O(n+m)$.

Beweis

DFS(G) 1. $O(n)$, 2. $O(n)$

ohne Zeit in DFS-v(u).

DFS-visit(u) $O(m)$ für Verwaltungsaufwand insgesamt.

DFS-visit(u) wird nur dann aufgerufen, wenn col[u] = weiß ist.

Am Ende des Aufrufs wird col[u] = schwarz.

1., 2. einmal $O(1)$ insgesamt $O(n)$

3., 4., 5., 6. ohne Zeit in DFS-v(v) insgesamt $O(m)$

(3., 4. für jede Kante einmal,
..., also $O(m)$
5., 6. für jedes Entdecken, $O(n)$)
7., 8. $O(n)$ insgesamt

Also tatsächlich $O(n+m)$ □

Definition

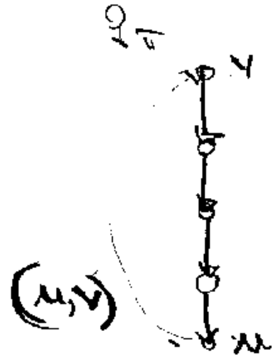
Sei $G = (V, E)$ und sei
DFS(G) gelaufen. Sei
$G_\pi = (V, E_\pi)$ mit
$(u,v) \in E_\pi \Leftrightarrow \pi[v] = u$
der Tiefensuchwald der Suche.

Klassifizieren der Kanten von G:
Sei $(u,v) \in E$.

(a) (u,v) Baumkante $\Leftrightarrow (u,v) \in E_\pi$
$(\pi[v] = u)$

(b) (u,v) Rückwärtskante

$\Leftrightarrow (u,v) \notin E_\pi$ und v Vorgänger von u in G_π



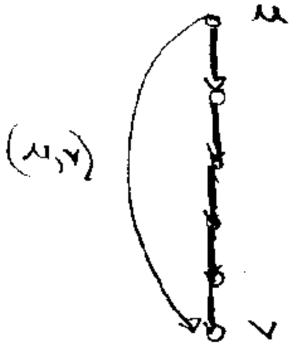
v grau, wenn (u,v) gegangen

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

Hier wird (u,v) gegangen.

(c) (u,v) Vorwärtskante

$\Leftrightarrow (u,v) \notin E_\pi$ und v Nachfolger von u in G_π



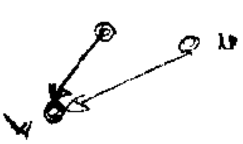
v schwarz, wenn (u,v) gegangen

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

Hier nicht (u,v) } Hier (u,v)

(d) (u,v) Kreuzkante

$\Leftrightarrow (u,v) \notin E_\pi$ und u weder Nachfolger noch Vorgänger von v im G_π



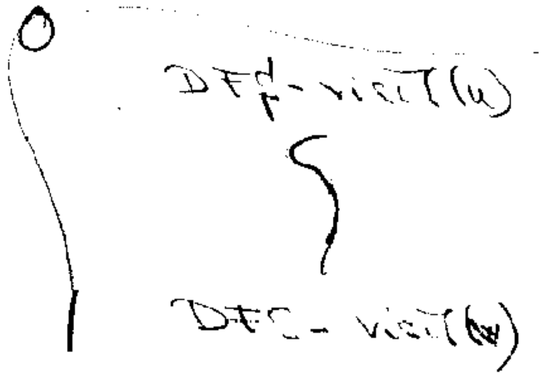
v schwarz wenn (u,v) gegangen

$d[v] < f[v] < d[u] < f[u]$ □

Noch eine Beobachtung für Knoten u, v. Für die Intervalle, in denen die Knoten aktiv (offen, grau) sind, gilt:

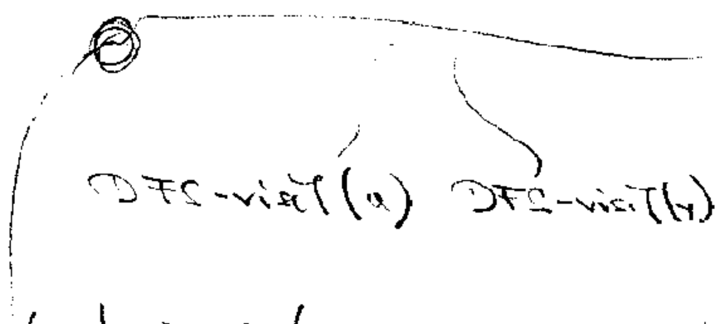
Entweder

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



oder

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



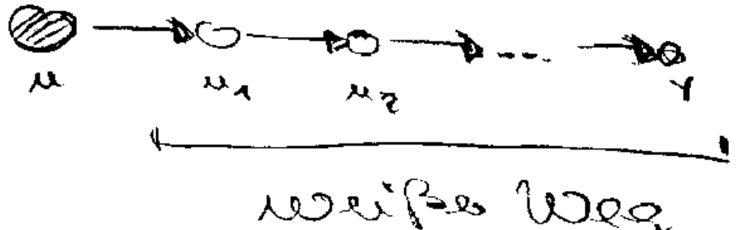
oder umgekehrt. Folgt aus Kellerstruktur des Laufzeitkellers.

Satz (Weiße Weg Satz)

v wird über u entdeckt (d.h. innerhalb von DFS-visit(u) wird DFS-visit(v) aufgerufen)

$\Longleftrightarrow$

Zum Zeitpunkt d[u] gibt es Weg



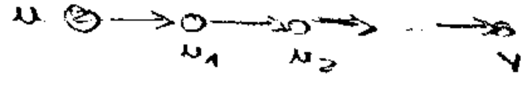
in G.

Beweis:

„$\Rightarrow$" Aufrufstruktur, wenn DFS-v(v) aufgerufen wird:

DFS-v(u)
|
DFS-v(u_1)
|
DFS-v(u_2)
⋮
DFS-v(v)

weißer Weg



„⇐“ Liegt also zu $d[u]$ der weiße Weg

$$u \to v_1 \to v_2 \to \cdots \to v_k \to v$$

vor. Das Problem ist, daß dieser Weg keineswegs von der Suche genommen werden muß. Trotzdem, angenommen v wird nicht über u entdeckt, dann nicht

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

sondern

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

Dann aber auch nach Programm

$$d[u] < f[u] < d[v_k] < f[v_k]$$

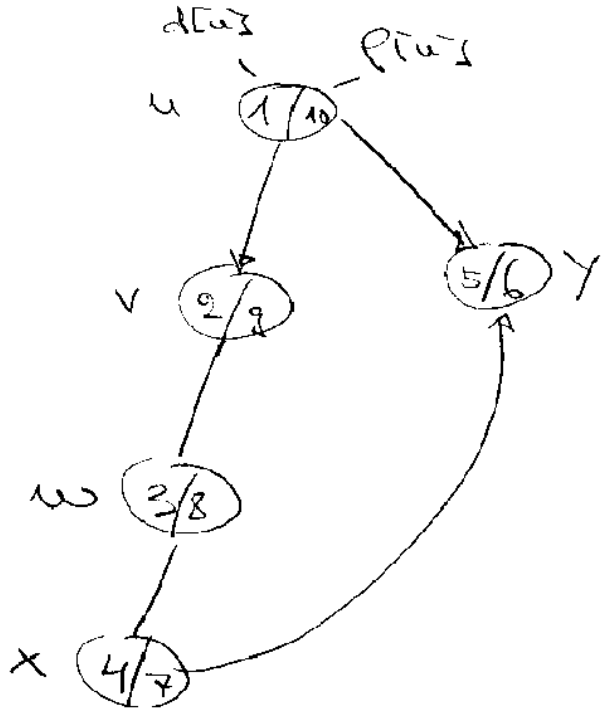
$$\vdots$$

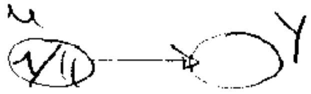
dann

$$d[u] < f[u] < d[v_1] < f[v_1]$$

was dem Programm widerspricht. □

Noch ein Beispiel



Zum Zeitpunkt 1 ist 

ein weißer Weg. Dieser wird nicht gegangen, sondern ein anderer. Aber y wird in jedem Falle über u entdeckt!

Kreise feststellen!

Satz

Ist $G = (V, E)$ gerichtet.

G hat Kreis

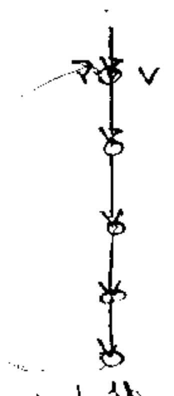
$\Leftrightarrow$

DFS(G) ergibt eine Rückwärtskante

Beweis

„$\Leftarrow$" Sei (u,v) eine Rückwärtskante.

Dann in G_π, dem Tiefensuchwald



oder auch

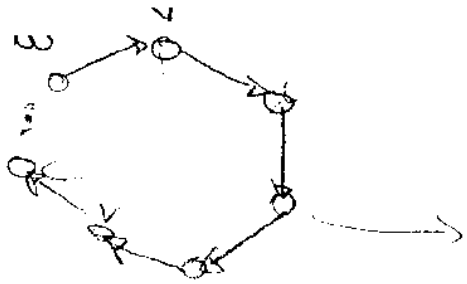
Rückwärtskante (u,v)

Rückwärtskante (u,v).

Also Kreis in G.

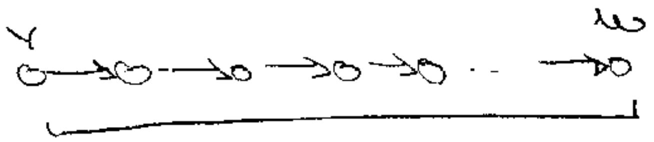
"⇒" Hat G einen Kreis, dann

also



Sei v der erste Knoten auf dem Kreis, den DFS(G) entdeckt.

Dann zu dem Zeitpunkt weißer Weg



Den Kreis herum.

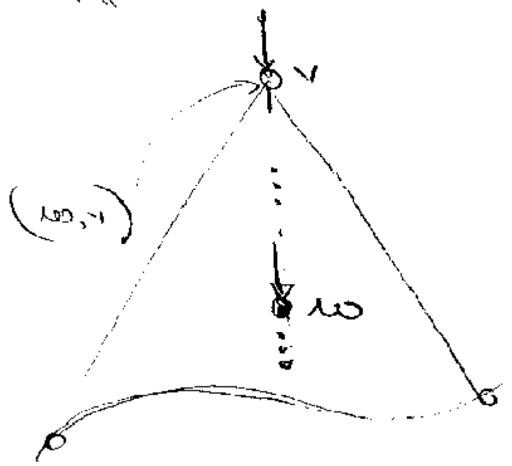
Weißer Weg Satz:

$d[v] < d[w] < f[w] < f[v]$

Also wenn (w,v) gegangen wird ist color[v] = grau also Rückwärtskante.

Alternativ, mit w-W-Satz.

In T_π



w irgendwo unter v und dann

(w,v) Rückwärtskante. □

Man vergleiche hier den Satz

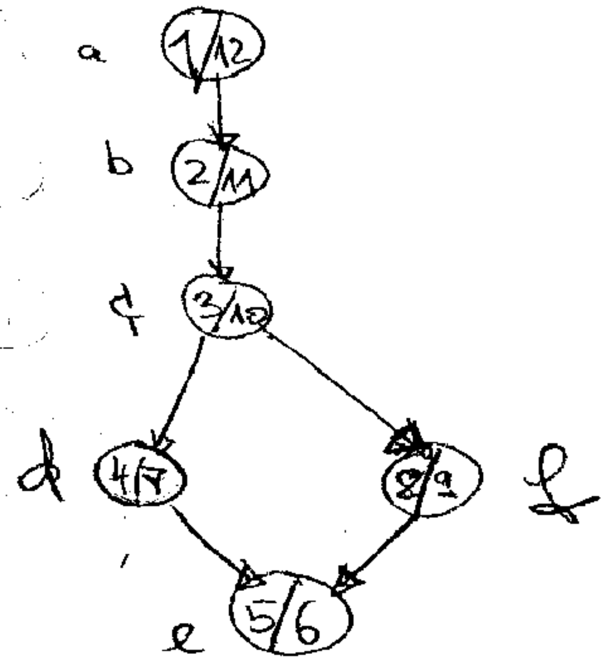
und den Beweis zu Kreisen in

ungerichteten Graphen in Kapitel 2.

Doch betrachte man den zuletzt (!)

auf dem Kreis entdeckten Knoten.

Wir können auch topologisch sortieren, wenn kreisfrei.



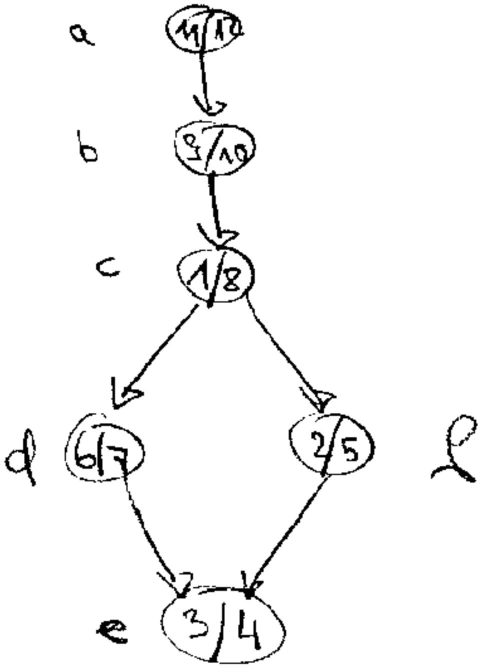
Nach absteigender Beendezeit sortieren

a, b, c, f, d, e

größte Beendezeit.

kleinste Beendezeit

Aber auch



Absteigende Beendezeit

a, b, c, d, f, e .

Satz

Sei $G = (V, E)$ kreisfrei. Lassen wir DFS(G) laufen, dann gilt für alle $u, v \in V$, $u \neq v$, daß

$$f[u] < f[v] \Rightarrow (u,v) \notin E.$$

keine Kante geht von kleinerer nach größerer Beendezeit.

Beweis

Statt $A \Rightarrow B$ direkt auch $\neg B \Rightarrow \neg A$.

Wir zeigen:

$$(u,v) \in E \Rightarrow f[u] \gneq f[v].$$

1. Fall: $d[u] < d[v]$

Dann w.Weg $u \to v$ zu $d[u]$,

also $d[u] < d[v] < f[v] \lneq f[u]$ wegen w-w-Satz.

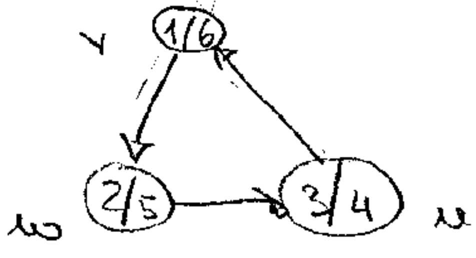
2. Fall $\quad d[u] > d[v]$

Zum Zeitpunkt $d[v]$ ist $col[u] = w$.
Aber da kreisfrei wird u nicht von v aus entdeckt, da sonst (u,v) Rückwärtskante und damit Kreis. Also kann nur sein

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

und es ist $f[v] \lneq f[u]$ ↯

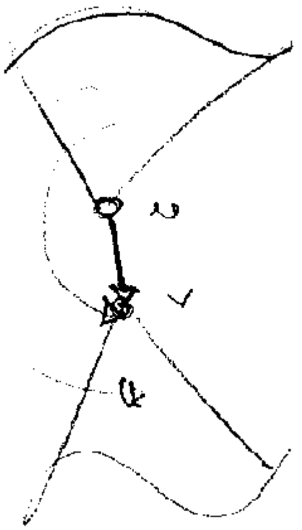
Beachte aber



$(u,v) \in E$ und $f[u] < f[v]$, der Satz gilt nicht. Aber haben ja auch einen Kreis!

$\neg(A \Rightarrow B)$ bedeutet $A \wedge \neg B$.

Im kreisfreien Fall ebenso so:



Im kreisfreien (!) Fall immer so:

$u \to v$

dann $\rho[v] < \rho[u]$.

Wird u vor v weiß dann klar $\rho[v] < \rho[u]$. Wird aber v vor u weiß dann wird u nicht Nachfolger von v also auch dann

$$\rho[v] \lneq \rho[u].$$

Wg kreisfrei.