Graphen: Erreichbarkeit, Durchsuchung

Gerichteter Graph $G = (V, E)$.
Welche andere Knoten können von Knoten s erreicht werden?
Was ist die “Zusammenhangsstruktur” von G?

- v von s *erreichbar* bedeutet, $\exists$ ein Weg von s nach v.

  $\exists$ Weg von s nach $v \Longrightarrow \exists$ einfacher Weg von s nach v (ohne Knotenwiederholungen)
- In dieser Vorlesung nur einfache Wege (betrachten “Weg” und “Pfad” synonym)
- Mit valider Kantenbeschriftung gibt es im Fall von Erreichbarkeit einen *eindeutigen* lex-kleinsten Weg $\mathrm{lkW}_s(v)$ von s nach v.

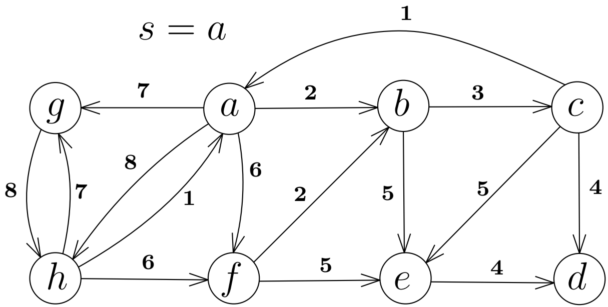
lex-kleinster-Wege-Baum

V_s sei Menge der von s erreichbaren Knoten

Die Menge aller lex-kleinsten Wege $\{\mathrm{lkW}_s(v) \mid v \in V_s\}$ bildet Baum mit Wurzel s, den *lex-kleinsten-Wege-Baum T_s von s.*

T_s ist ein aufspannender Baum von V_s.

$s = a$

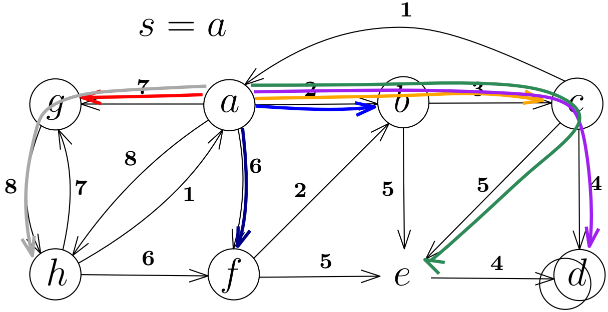


lex-kleinster-Wege-Baum

V_s sei Menge der von s erreichbaren Knoten

Die Menge aller lex-kleinsten Wege $\{\mathrm{lkW}_s(v) \mid v \in V_s\}$ bildet Baum mit Wurzel s, den *lex-kleinsten-Wege-Baum T_s von s.*

T_s ist ein aufspannender Baum von V_s.

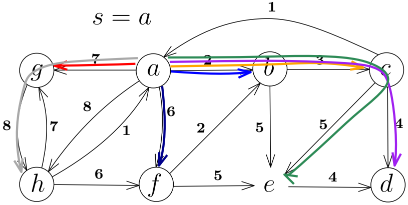


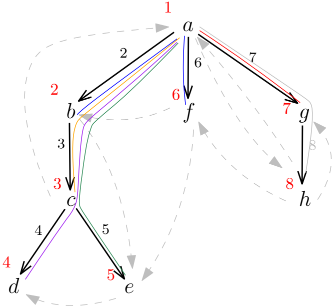
lex-kleinster-Wege-Baum

V_s sei Menge der von s erreichbaren Knoten

Die Menge aller lex-kleinsten Wege $\{\mathrm{lkW}_s(v) \mid v \in V_s\}$ bildet Baum mit Wurzel s, den *lex-kleinsten-Wege-Baum T_s von s.*

T_s ist ein aufspannender Baum von V_s.



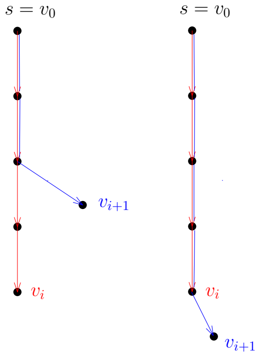


Konstruktion des lex-kleinsten-Wege-Baums von s

Die Knoten in V_s werden durch die lexikographische Ordnung der Beschriftungen der Wege ein eine Reihenfolge gesetzt:
$\mu(\mathrm{lkW}_s(v_1)) \prec \mu(\mathrm{lkW}_s(v_2)) \prec \cdots \prec \mu(\mathrm{lkW}_s(v_{|V_s|}))$
mit $v_1 = s$ und $\mu(\mathrm{lkW}_s(v_1)) = \varepsilon$ (der leere String)
Idee: Bestimme für jeden von s erreichbaren Knoten v den Weg $\mathrm{lkW}_s(v)$, und zwar in der Reihenfolge $v_1, v_2, \ldots$. (Dadurch wird auch V_s gefunden)

Wie unterscheiden sich $\mathrm{lkW}_s(v_i)$ und $\mathrm{lkW}_s(v_{i+1})$?

$\mathrm{lkW}_s(v_{i+1})$ besteht aus einem Präfix von $\mathrm{lkW}_s(v_i)$ und einer weiteren Kante.



Konstruktion des lex-kleinsten-Wege-Baums von s

lkW-VISIT(s) findet diesen Baum, indem die von s erreichbaren Knoten in lex-Ordnung der lex-kleinsten Wege aufgezählt werden. $r[v]$ ist der Rang von v in dieser Ordnung (und auch die Präordernummer des Knotens im gefundenen Baum. $\pi[v]$ ist der Elterknoten in dem Baum

Initialisierung für alle v: $r[v] = \mathrm{undef}$
Globale Variable: zr=0

```
lkW-VISIT(Knoten v): ;                          // Annahme v noch unbekannt
r[v] = ++zr
for jede Kante [v, w⟩ die v verlässt in Beschriftungsordnung
 do
    if w noch unbekannt then
        π[w] = v ;                               // Elterzeiger setzen
        lkW-VISIT(w)
```

Konstruktion des lex-kleinsten-Wege-Baums von s

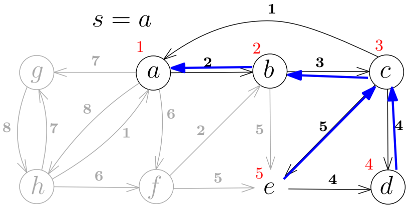
lkW-VISIT(s) findet diesen Baum, indem die von s erreichbaren Knoten in lex-Ordnung der lex-kleinsten Wege aufgezählt werden. $r[v]$ ist der Rang von v in dieser Ordnung (und auch die Präordernummer des Knotens im gefundenen Baum. $\pi[v]$ ist der Elterknoten in dem Baum

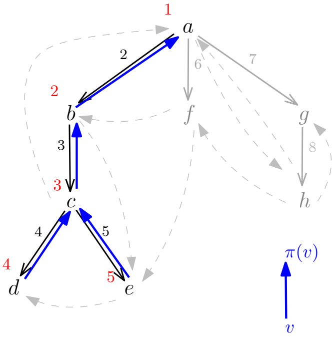
Initialisierung für alle v: $r[v] = \mathrm{undef}$
Globale Variable: zr=0

lkW-VISIT(Knoten v): ; // Annahme v noch unbekannt
$r[v] = {+}{+}zr$
for *jede Kante* $[v, w\rangle$ *die* v *verlässt in Beschriftungsordnung* **do**
    if ~~*w noch unbekannt*~~ $r[w] = \mathit{undef}$ **then**
        $\pi[w] = v$; // Elterzeiger setzen
        lkW-VISIT(w)

Konstruktion des lex-kleinsten-Wege-Baums von *s*

Beispiel:





Zwischenergebnis nach Folge von Aufrufen:

lkW-VISIT(a)
  kW-VISIT(b)
    lkW-VISIT(c)
      lkW-VISIT(d)
      lkW-VISIT(e)

Konstruktion des lex-kleinsten-Wege-Baums von *s*

Mit lkW-VISIT() können auch die Postordnungnummern der Knoten im lex-kleinsten-Wege-Baum gefunden werden.

Initialisierung für alle v**:** $r[v] = \mathrm{undef}$; $p[v] = \mathrm{undef}$
Globale Variablen: zr=0; zp=0

```
lkW-VISIT(Knoten v): ;                      // Annahme v noch unbekannt
r[v] = ++zr
for jede Kante [v, w⟩, die v verlässt, in
  Beschriftungsordnung do
    if w noch unbekannt r[w] = undef then
        π[w] = v ;                          // Elterzeiger setzen
        lkW-VISIT(w)
p[v] = ++zp
```

Konstruktion des lex-kleinsten-Wege-Baums von s

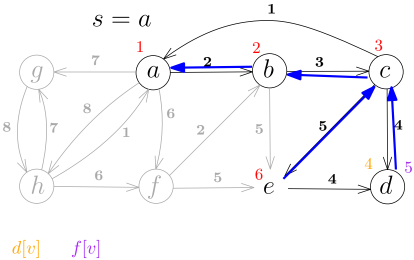
Alternativ (oder zusätzlich) können auch die $d[v]$ ("discovery")und $f[v]$ ("finish") des lex-kleinsten-Wege-Baum gefunden werden .

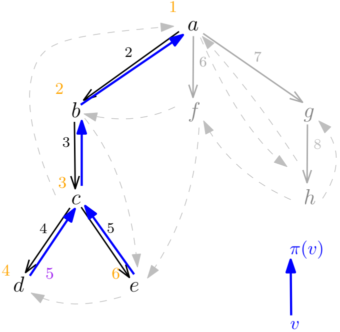
Initialisierung für alle v: $d[v] = \text{undef}$; $f[v] = \text{undef}$
Globale Variablen: zdf=0;

lkW-VISIT(Knoten v): ; // Annahme v noch unbekannt
$d[v] = {+}{+}zdf$
for *jede Kante* $[v, w\rangle$*, die v verlässt, in Beschriftungsordnung* **do**
    if ~~*w noch unbekannt*~~ $d[w] = \mathit{undef}$ **then**
        $\pi[w] = v$; // Elterzeiger setzen
        lkW-VISIT(w)
$f[v] = {+}{+}zdf$

Konstruktion des lex-kleinsten-Wege-Baums von s

Beispiel:



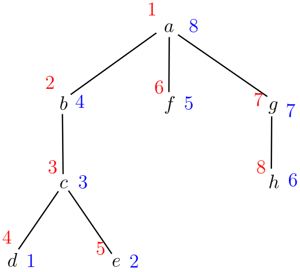


Zwischenergebnis nach Folge von Aufrufen:

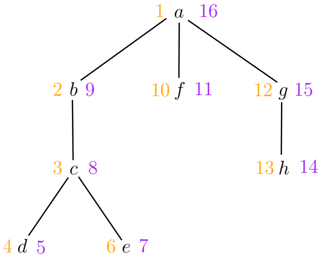
lkW-VISIT(a)
kW-VISIT(b)
lkW-VISIT(c)
lkW-VISIT(d)
lkW-VISIT(e)

Kantenklassifikation in lex-kleinsten-Wege-Bäumen

Zeichne Bäume geordnet: Wurzel oben, Kinder darunter, von links nach rechts nach Kantenbeschriftung geordnet. Die Kinder sind dann auch jeweils nach *r*[], *p*[], *d*[] und *f*[] geordnet.



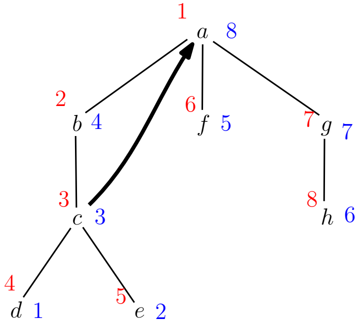
Präordnungsnummerierun
Postordnungsnummerieru



d-f Nummerierung

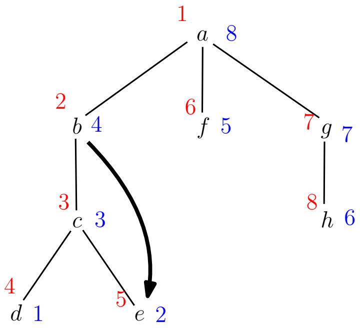
Kantenklassifikation in lex-kleinsten-Wege-Bäumen

$[u, v\rangle$ *Aufkante*: v ist Ahne von u



Kantenklassifikation in lex-kleinsten-Wege-Bäumen

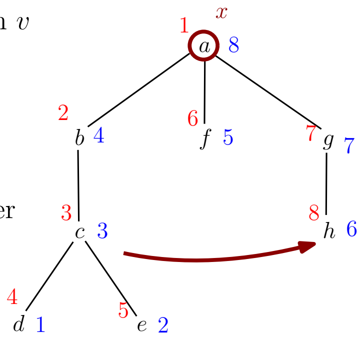
$[u, v\rangle$ *Abkante*: v ist Nachkomme von u



Kantenklassifikation in lex-kleinsten-Wege-Bäumen

x nächster gemeinsame Ahne von u unn v

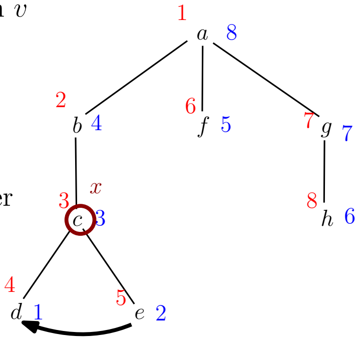
$[u, v\rangle$ *Rechtskante*:
der Unterbaum von x mit v ist weiter rechts als der Unterbaum mit u



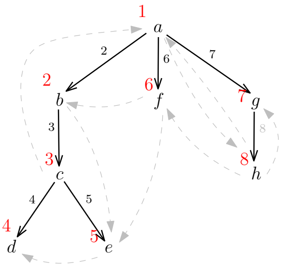
Kantenklassifikation in lex-kleinsten-Wege-Bäumen

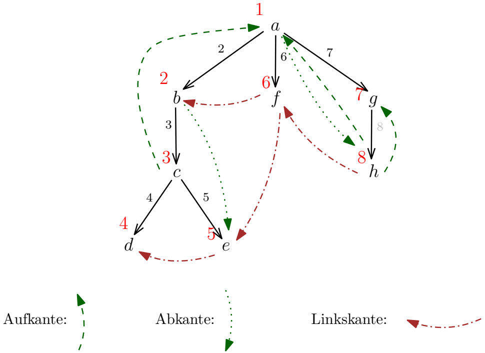
x nächster gemeinsame Ahne von u unn v

$[u, v\rangle$ *Linkskante*:
der Unterbaum von x mit v ist weiter links als der Unterbaum mit u

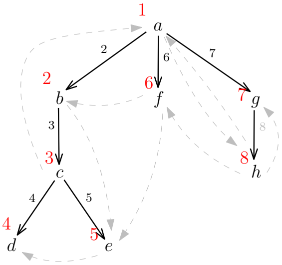


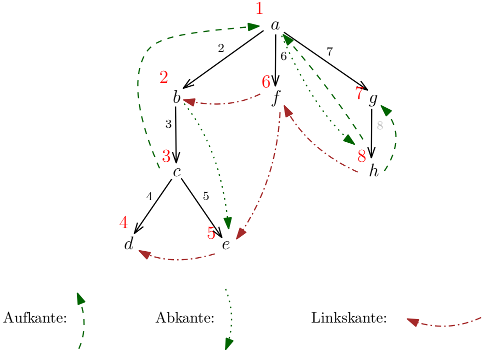
Kantenklassifikation in lex-kleinsten-Wege-Bäumen





Kantenklassifikation in lex-kleinsten-Wege-Bäumen





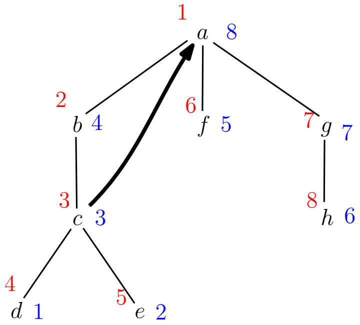
In einem lex-kleinsten-Wege-Baum kann es keine Rechtskanten geben.

Das würde dem Nachkommenlemma widersprechen.

Kantenklassifikation in lex-kleinsten-Wege-Bäumen und Knotennummerierungen

$[u, v\rangle$ *Aufkante*: v ist Ahne von u

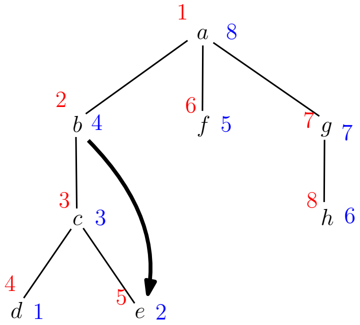
$r[v] < r[u]$ und $p[v] > p[u]$



Kantenklassifikation in lex-kleinsten-Wege-Bäumen und Knotennummerierungen

$[u, v\rangle$ *Abkante*: v ist Nachkomme von u

$r[v] > r[u]$ und $p[v] < p[u]$

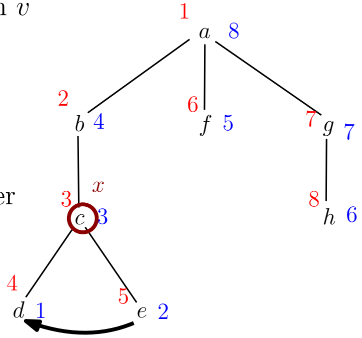


Kantenklassifikation in lex-kleinsten-Wege-Bäumen und Knotennummerierungen

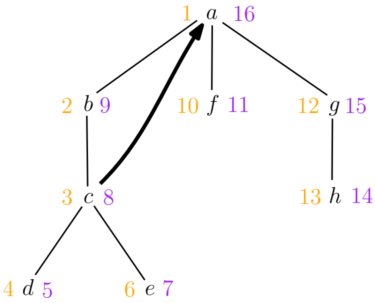
x nächster gemeinsame Ahne von u unn v

$[u, v\rangle$ *Linkskante*:
der Unterbaum von x mit v ist weiter links als der Unterbaum mit u

$r[v] < r[u]$ und $p[v] < p[u]$



Kantenklassifikation in lex-kleinsten-Wege-Bäumen und $d[]$ – $f[]$ Knotennummerierungen



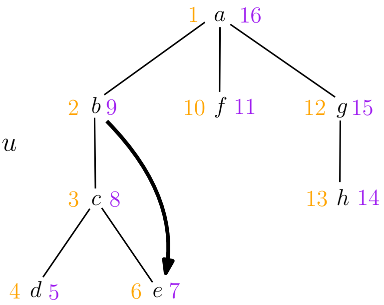
$[u, v\rangle$ *Aufkante*: v ist Ahne von u

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

Intervall $I[u] = [d[u], f[u]]$ ist im
Intervall $I[v] = [d[v], f[v]]$ enthalten

Kantenklassifikation in lex-kleinsten-Wege-Bäumen und $d[]$ – $f[]$ Knotennummerierungen

$[u, v\rangle$ *Abkante*: v ist Nachkomme von u

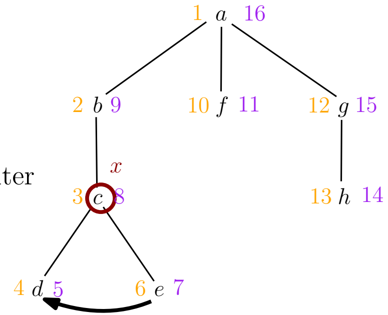


$d[v] > d[u]$ und $f[v] < f[u]$

Intervall $I[u] = [d[u], f[u]]$ enthält
das Interval $I[v] = [d[v], f[v]]$

Kantenklassifikation in lex-kleinsten-Wege-Bäumen und $d[]$ – $f[]$ Knotennummerierungen

x nächster gemeinsame Ahne von u unn v



$[u, v\rangle$ *Linkskante*:
der Unterbaum von x mit v ist weiter links als der Unterbaum mit u

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

Intervall $I[u] = [d[u], f[u]]$ rechts vom Intervall $I[v] = [d[v], f[v]]$ auf der Zahlenlinie

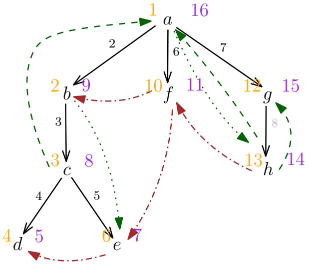
Kantenklassifikation in lex-kleinsten-Wege-Bäumen und $d[]$ – $f[]$ Knotennummerierungen

$[u, v\rangle$ Abkante $\iff I[v] \subset I[u]$

$[u, v\rangle$ Aufkante $\iff I[v] \supset I[u]$

$[u, v\rangle$ Linkskante $\iff I[v]$ links von $I[u]$ auf der Zahlengerade

Intervall $I[w] = [d[w], f[w]]$



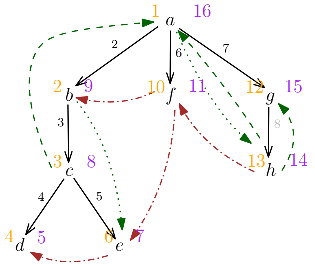
Aufkante: Abkante: Linkskante:

Kantenklassifikation in lex-kleinsten-Wege-Bäumen und *d*[] – *f*[] Knotennummerierungen

$[u, v\rangle$ Abkante $\iff I[v] \subset I[u]$

$[u, v\rangle$ Aufkante $\iff I[v] \supset I[u]$

$[u, v\rangle$ Linkskante $\iff I[v]$ links von $I[u]$ auf der Zahlengerade



Intervall $I[w] = [d[w], f[w]]$

Aufkante:

Abkante:

Linkskante:

Die I[] Intervalle verschiedener Knoten sind entweder verschachtelt oder disjunkt.

Anwendungen von Knotennummerierungen

$G = (V, E)$ gerichteter Graph und T_s ein lex-kleinster-Wege-Baum von G (s erreicht alle Knoten in V).

Lemma: G enthält einen Zyklus genau dann wenn es in T_s eine Aufkante gibt.

Beweis: "$\Leftarrow$" offensichtlch

"$\Rightarrow$' Aufgabe 3 auf dem letzten Übungsblatt (betrachte Zyklus C und v den lex-kleinsten Knoten auf C; v hat Vorgänger u auf C; wegen Nachkommenlemma muss u ein Nachkomme von v sein, also ist $[u, v\rangle$ eine Aufkante)

Anwendung: Testen, ob G azyklisch:
Augmentiere G mit $\diamond$ und, lkW-VISIT($\diamond$) und überprüfe, ob eine der Kanten eine Aufkante ist. (Laufzeit $O(n + m)$)
Wenn G azyklisch ist, dann bilden die $f[]$-Nummern in umgekehrter Reihenfolge eine topologische Sortierung.

(wenn $[u, v\rangle$ eine Abkante oder eine Linkskante ist, dann gilt $f[u] > f[v]$. Andere Arten von Kanten gibt es in diesem Fall nicht.)

DFS (Tiefensuche) und lex-kleinste-Wege-Bäume

Initialisierung für alle v**:** $d[v] = \text{undef}$; $f[v] = \text{undef}$
Globale Variablen: zdf=0;

1 lkW-VISIT(Knoten v): ; // Annahme v noch unbekannt
2 $d[v] = {+}{+}zdf$
3 **for** *jede Kante* $[v, w\rangle$*, die* v *verlässt,*
in Beschriftungsordnung do
4 **if** ~~*w noch unbekannt*~~ $d[w] = \mathit{undef}$ **then**
5 $\pi[w] = v$; // Elterzeiger setzen
6 lkW-VISIT(w)
7 $f[v] = {+}{+}zdf$

Man kann auch so tun, als würden die Beschriftungen der Kanten erst während des Durchlaufens der Schleife von Zeile 3 festgelegt — und zwar in der Reihenfolge, in der diese Kanten, die v verlassen, eben behandelt werden.
Da die Kantenbeschriftungen *algorithmisch* sonst keine Rolle spielen (für mathematische Beweise aber schon) kann man sie auch vollkommen weglassen. Der sich so ergebende Algorithmus ist als *Tiefensuche* oder *Depth-First-Search (DFS)* bekannt.

DFS (Tiefensuche) und lex-kleinste-Wege-Bäume

Initialisierung für alle v: $d[v] = \mathrm{undef}$; $f[v] = \mathrm{undef}$
Globale Variablen: zdf=0;

DFS-VISIT(Knoten v): ; // `Annahme v noch unbekannt`
$d[v] = {++}zdf$
for *jede Kante* $[v, w\rangle$*, die v verlässt,* **do**
  if ~~*w noch unbekannt*~~ $d[w] = \mathit{undef}$ **then**
    $\pi[w] = v$; // `Elterzeiger setzen`
    DFS-VISIT(w)
$f[v] = {++}zdf$

DFS (Tiefensuche) und lex-kleinste-Wege-Bäume

Wenn $G = (V, E)$ mit $\diamond$ augmentiert wird, dann gilt $\mathrm{Out}(\diamond) = V$, d.h. ein Aufruf DFS-VISIT($\diamond$) entspricht:

DFS($G = (V, H)$)
Initialisierung für alle v**:** $d[v] = \mathrm{undef}$; $f[v] = \mathrm{undef}$
Globale Variablen: zdf=0;
for *jeden Knoten* $w \in V$ **do**
    if ~~*w noch unbekannt*~~ $d[w] = \mathit{undef}$ **then**
        DFS-VISIT(w)

DFS (Tiefensuche) und lex-kleinste-Wege-Bäume

Wenn $G = (V, E)$ mit $\diamond$ augmentiert wird, dann gilt $\mathrm{Out}(\diamond) = V$, d.h. ein Aufruf DFS-VISIT($\diamond$) entspricht:

DFS($G = (V, H)$)
Initialisierung für alle v**:** $d[v] = \text{undef}$; $f[v] = \text{undef}$
Globale Variablen: zdf=0;
for *jeden Knoten* $w \in V$ **do**
  if ~~*w noch unbekannt*~~ $d[w] = \mathit{undef}$ **then**
    DFS-VISIT(w)

Die Laufzeit ist $O(n + m)$ mit $n = |V|$ und $m = |E|$, da jede Kante nur einmal betrachtet wird.

Anwendungen von DFS – Zusammenhangskomponenten

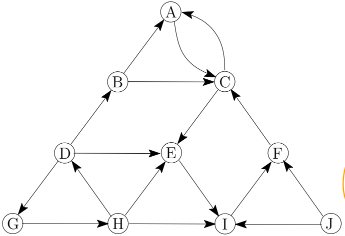
Wenn $G = (V, E)$ ein **ungerichteter** Graph ist, dann git es im lex-kleinsten-Wege Baum, bzw. im DFS-Baum nur Aufkanten und Abkanten. Linkskanten nicht möglich, da es dazu die symmetrischen Rechtskanten geben müsste.

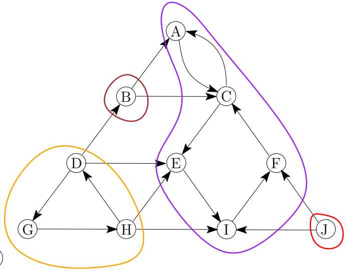
In einem ungerichteten Graphen ist Erreichbarkeit zwischen zwei Knoten eine Äquivalenzrelation. Die Äquivalenzklassen heißen Zusammenhangskomponenten.

DFS($G = (V, E)$) bestimmt die Zusammenhangskomponenten in linearer Zeit: jedesmal, wenn in Zeile 6 DFS-VISIT(w) aufgerufen wird, wird der DFS-Baum und damit die Bestimmung einer weiteren Zusammenhangskomponente begonnen.

Anwendungen von DFS – starke Zusammenhangskomponenten

In einem **gerichteten** Graphen $G = (V, E)$ ist gegenseitige Erreichbarkeit zwischen Knoten eine Äquivalzenrelation (also u und v äquivalent, wenn es einen Weg von u nach v und einen Weg von v nach u gibt).
Die Äquivalenzklassen dieser Relation heißen die *starken Zusammenhangskomponenten* von G.





Berechnung von starken Zusammenhangskomponenten: Methode 1

(von Kosaraju und von Sharir, nicht in Vorlesung behandelt)

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

1. Führe DFS($G = (V, E)$) durch und berechne die $f[]$-Werte.
2. Bilde den Graphen $G^T = (V, E^T)$, wobei E^T alle Kanten aus E enthält, aber mit umgedrehter Orientierung, also aus $[u, v\rangle$ wird $[v, u\rangle$. Augmentiere G^T und gib jeder Kante $[\diamond, v\rangle$ die Beschriftung $-f[v]$ (die anderen Kantenbeschriftungen sind beliebig, aber valide).
3. Führe lkW-VISIT($\diamond$) auf G^T aus und berechne Baum $T_\diamond$. Jedes Kind v von $\diamond$ in $T_\diamond$ ergibt durch die Knoten des bei ihm verwurzelten Teilbaums eine starke Zhgskomponente.

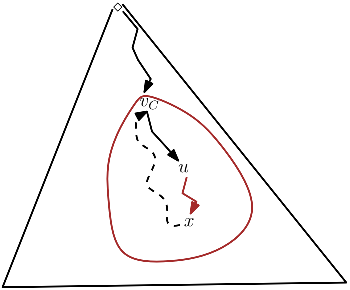
Laufzeit ist linear, da jeder der 3 Schritte diese Laufzeit hat.

Berechnung von starken Zusammenhangskomponenten: Methode 0

(nach Tarjan)

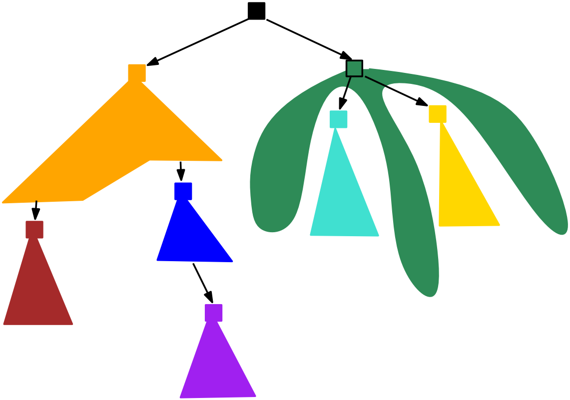
Lemma: Es sei $T_\diamond$ ein lex-kleinster-Wege Baum für augmentierten gerichteten Graphen $G_\diamond$ und sei C eine starke Zusammenhangskomponente von G.
Die Knoten in C bilden einen (zusammnehängenden) Teilbaum von $T_\diamond$.

Beweis: Sei v_C der der lex-kleinste Knoten in C und x irgendein Knoten in C. Wegen des Nachfolgerlemmas muss x ein Nachfolger von v_C sein in $T_\diamond$ sein. Wir müssen zeigen, dass jeder Knoten u auf dem Weg P in $T_\diamond$ zwischen v_C und x auch in C liegt. Offensichtlich erreicht v_C den Knoten u entlang P. Knoten u erreicht auch v_C weil er entlang P den Knoten x erreicht, und dieser, weil in C, auch v_C erreichen kann.



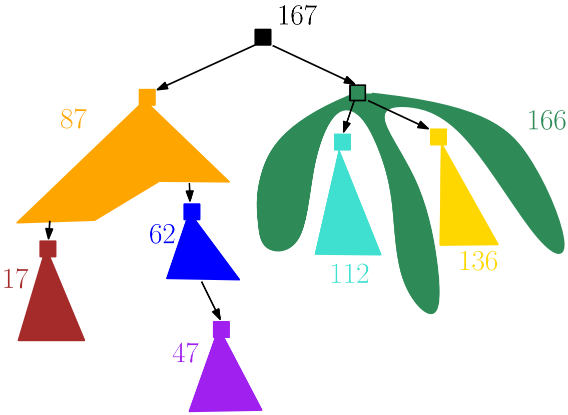
Starke Zusammenhangskomponenten

Nennen wir den lex-kleinsten Knoten v_C einer starken Zusammenhangskomponente C das *Tor* von C.
Man kann sich die Menge der Zusammenhangskomponenten mit ihren Toren als baumartige Struktur vorstellen.



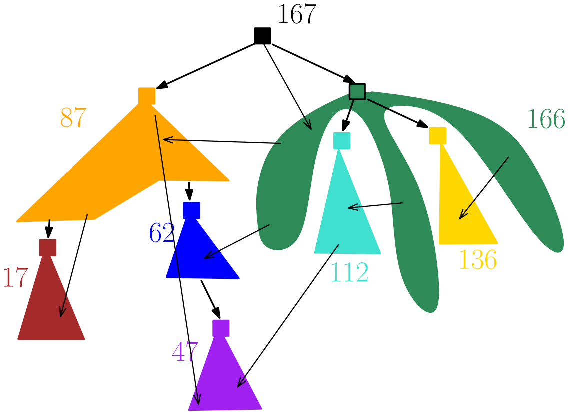
Starke Zusammenhangskomponenten

Nummeriere die Komponenten nach der $f[]$-Nummer ihres jeweiligen Tores.



Starke Zusammenhangskomponenten

Nummeriere die Komponenten nach der $f[]$-Nummer ihres jeweiligen Tores. Das ergibt eine Postordnung in der baumartigen Struktur. Kanten von G zwischen den Komponenten können nur von größerer zu kleinerer Nummer führen. Da $T_\diamond$ keine Rechtskanten hat, wäre eine Kante von kleiner zu großer $f[]$-Nummer eine Aufkante, was die beiden Komponenten zu einer machen würde.



Starke Zusammenhangskomponenten

Ziel: Weise jedem Knoten $v \in V$ die $f[]$-Nummer des Tores seiner starken Zusammenhangskomponente zu, sagen wir in Variable $K[v]$, die anfangs auf undef gesetzt ist.

Strategie: Führe lkW-Suche durch und identifiziere alle Tore in Reihenfolge aufsteigend in ihrer $f[]$-Nummer. Wenn ein Tor v identifiziert wird, dann ist die dazugehörige Zusammenhangskomponente ein Teilbaum des lkW-Baums mit Wurzel v und der besteht aus allen Nachkommen von v, denen noch kein $K[]$ Wert zugewiesen wurde.

Starke Zusammenhangskomponenten

Algorithmus, um alle Knoten in Zusammenhangskomponente mit Wurzel v mit Zahl α zu markieren.

Annahmen: Die relevanten Teile des lkW-Baums sind schon berechnet, also für alle x mit $f[x] \leq f[v]$ sind $d[x]$ und $f[x]$ und $\pi[x]$ bekannt, und für alle x mit $d[x] \leq d[v]$ sind $d[x]$ und $\pi[x]$ bekannt. Außerdem sind die Knoten der Zusammenhangskomponenten mit kleinerer Nummer schon entsprechend markiert (K[]-Wert zugewiesen).

MarkiereZHK(Knoten v, Wert α): ; // `Annahme` v `unmarkiert`
$K[v] = \alpha$
for *jede Kante* $[v, w\rangle$*, die* v *verlässt,* **do**
  if $\pi[w] = v$ *und* $K[w] = \mathit{undef}$ **then**
    $K[w] = \alpha$
    MarkiereZHK(w, α)

Laufzeit ist proportional zu $\sum_{w \text{ in markierter Komponente}} \mathrm{outdeg}(w)$.

Toridentifizierung

v ist **kein** Tor, wenn es einen Weg P von einem Nachkommen w zu einem echten Ahnen u von v gibt. Es sei $[x, y\rangle$ eine Kante auf P, die den Unterbaum von v verlässt, also x ist Nachkomme von v aber y nicht.

Möglichkeit 1: y ist echter Ahne von v

Dann gilt $d[y] < d[v]$

Möglichkeit 1: y ist kein Ahne von v

Dann ist $[x, y\rangle$ eine Linkskante und damit $d[y] < d[v]$.

In beiden Fällen liegt y in der gleichen Zusammenhangskomponente wie v, also nicht in einer schon gefundenen.

Definiere $\mathrm{lowlink}(v)$ als das Minimum von $d[v]$ und dem kleinsten $d[y]$-Wert über alle Kanten $[x, y\rangle$ mit x im Unterbaum von v und mit y in noch keiner bekannten Zusammenhangskomponente.

Beobachtung: v ist ein Tor $\iff \mathrm{lowlink}(v) = d[v]$

Berechnung starke Zusammenhangskomponenten

Gegeben Augmentierung von gerichtetem Graph $G = (V, E)$

Initialisierung für alle v**:** $d[v] = f[v] = K[v] = \text{undef}$
Globale Variablen: zdf=0;

SZK($\diamond$) wobei

SZK(Knoten v) // Annahme v noch unbekannt
$d[v] = ++zdf$
$\text{lowlink}[v] = d[v]$
for *jede Kante* $[v, w\rangle$, *die* v *verlässt* **do**
    if ~~w noch unbekannt~~ $d[w] = \mathit{undef}$ **then**
        $\pi[w] = v$ // Elterzeiger setzen
        SZK(w)
        $\text{lowlink}[v] = \min\{\text{lowlink}[v], \text{lowlink}[w]\}$
    else
        if $K[w] = \mathit{undef}$ **then**
            $\text{lowlink}[v] = \min\{\text{lowlink}[v], d[w]\}$
$f[v] = ++zdf$
if $\mathit{lowlink}[v] = d[v]$ **then**
    MarkiereZHK(v, $f[v]$)

Berechnung starke Zusammenhangskomponenten: Laufzeit

SZK($\diamond$) braucht so viel Zeit wie DFS (oder lkW-visit($\diamond$)), also $O(n + m)$ plus die Zeiten für die MarkiereZHK() Aufrufe. Da aber jeder Knoten in genau einer starken Zusammenhangskomponente liegt, ist die Zeit dafür insgesamt die Summe aller Knoten-Outdegrees, also auch $O(n + m)$.