DFS, BFS

Exploration eines Digraphen von einem Startknoten aus

Eingabe: Digraph $G = (V, E)$, Startknoten $s \in V$

Datenstrukturen:

- G in Adjazenzlistendarstellung, L_v = Nachbarschaftsliste des Knoten v
- Markierung der Knoten als „neu" (noch unbesucht) oder „alt" (bereits besucht)
- Liste L bereits besuchter Knoten, deren Nachbarschaftsliste noch nicht vollständig durchforstet wurde.

Methode:

1. $L := \{s\}$; markiere s „alt".
2. Markiere alle $v \in V \setminus \{s\}$ „neu".
3. Wiederhole folgende Schritte bis L leer ist:
   (a) Wähle einen Knoten v aus L.
   (b) **Fall 1:** v besitzt einen „neu" markierten Nachbarn $w \in L_v$
   Füge w in L ein und markiere w „alt".
   Fall 2: – sonst –
   Entferne v aus L.

Bem.: 1) Bei konkreten Anwendungen werden an diesem Code weitere verarbeitungsschritte eingefügt.

2) Die Methode „Depth First Search (DFS)" entsteht, wenn L als STACK behandelt wird,

3) Die Methode „Breadth First Search (BFS)" entsteht, wenn L als QUEUE behandelt wird.

4) Wenn von s aus nicht alle Knoten über Pfade erreicht werden können, muss obige Methode mehrmals (mit jeweils neuen Start knoten) angewendet werden.

Def: Wenn von s aus alle Knoten in V über Pfade in G erreichbar sind, heißt s **Wurzelknoten** in G.

DFS

```
algorithm dfs(v)                                   } Prozedur für
  if v ist „neu"                                   } 1 Durchlauf;
    then markiere v als „alt";                     }
      for each w ∈ L_v do dfs(w) od
  fi
```

```
Im Hauptprogramm   V = [n] = {1, ..., n}          } Initialisierung
for v := 1 to n do visited[v] := false od ;        } +
for v := 1 to n do dfs(v) od .                     } Neustarts
```

Die Behandlung als STACK erfolgt hier implizit durch die rekursive Prozedur dfs.

visited[v] = true ≙ v ist „alt"
visited[v] = false ≙ v ist „neu"

BFS

Algorithm bfs(v)

1) Füge in die leere QUEUE q
   den Knoten v ein;
   Markiere v als „alt".
2) **while** q ist nicht leer **do**
   Entnimm q das erste Element,
   sagen wir w;
   for each $w' \in L_w$ **do**
   if w' ist „neu"
   then
   Füge w' (hinten) in q ein;
   markiere w' als „alt"
   endif
   endfor
   endwhile

Prozedur für 1 Durchlauf

Im Hauptprogramm:

for v := 1 **to** n **do** visited[v] := false **od**; (markiere v als „neu")

for v := 1 **to** n **do**
if v ist „neu" **then** bfs(v) **endif** **od**

Satz: Sei $n = |V|$, $m = |E|$.
DFS bzw. BFS benötigt nur $O(n+m)$ Rechenschritte

Grund: Bei geeigneter Implementierung werden die Adjazenzlisten nur 1-mal durchsucht.

BFS- und DFS-Wald

↱ geeignet erweitert

BFS angewendet auf $G = (V, E)$

$\longrightarrow$ BFS-Wald von G,

↱ geeignet erweitert

DFS angewendet auf $G = (V, E)$

$\longrightarrow$ DFS-Wald von G,

beides (auf-)Spannende Wälder von G.

} Weiteres in Olntm) Rechenschnitten

Def:

1) $[T_1, \dots, T_k] = [(V_1, E_1), \dots, (V_k, E_k)]$

heißt **Spannender Wald** von $G = (V, E)$

$:\Leftrightarrow$

(i) $[V_1, \dots, V_k]$ ist eine Partition von V

(ii) $\forall i = 1, \dots, k$:

$T_i = (V_i, E_i)$ ist ein Baum,

der zu dem Teilgraph von G ist.

2) Im Falle $k = 1$ sprechen wir von einem **Spannenden Baum** von G.

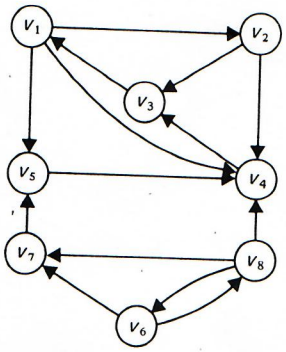
Formale Def. von BFS- bzw. DFS-Wald

a) Knoten, an denen der Durchlauf neu gestartet wird, werden zu Wurzeln

b) Wenn Knoten w von Knoten v aus zum ersten Mal besucht wird, wird w der Sohn von v.

} kann leicht in den DFS-, BF-Code integriert werden

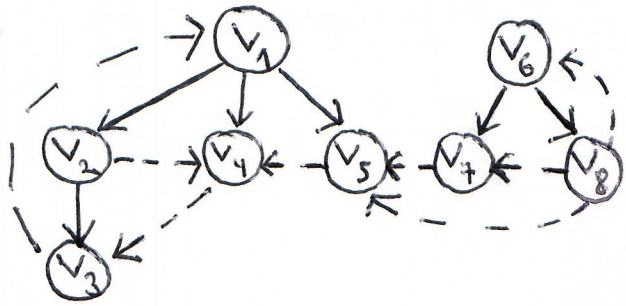
Beispiel:



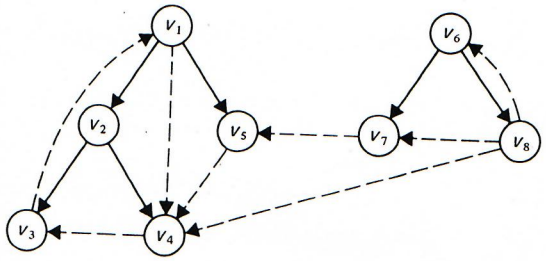
} G

Söhne eines Knoten v können in der Reihenfolge, in der sie besucht werden, von links nach rechts gezeichnet werden.

BFS-Wald von G



DFS-Wald von G



Nicht zum Wald gehörende Kanten aus E sind gestrichelt gezeichnet!

Typologie für Kanten aus $E \setminus \bigcup_{i=1}^{k} E_i$

Notation: $W = [T_1, \dots, T_k]$ sei spannender Wald von G. $x, y \in V$

$x \xrightarrow[W]{} y \quad :\Leftrightarrow$ y ist Sohn von x in W

$x \xrightarrow[W]{*} y \quad :\Leftrightarrow$ y ist Nachkomme von x in W

$x \xrightarrow[W]{+} y \quad :\Leftrightarrow$ y ist echter Nachkomme von x in W

Def: 1) Eine Kante $e = (x, y)$ aus $E \setminus \bigcup_{i=1}^{k} E_i$ heißt

Foward-Edge (F-Kante) $:\Leftrightarrow$ $x \xrightarrow[W]{+} y$

Backward-Edge (B-Kante) $:\Leftrightarrow$ $y \xrightarrow[W]{+} x$

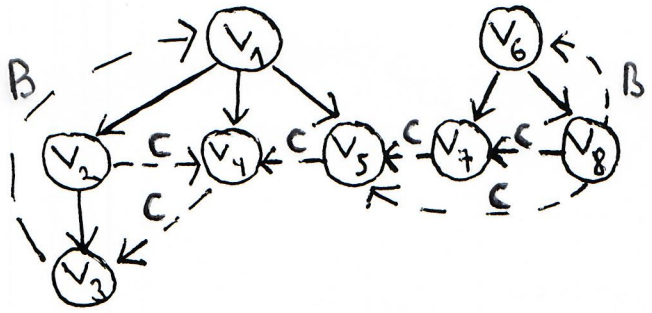
Crossing-Edge (C-Kante), sonst.

2) Die Kanten aus $\bigcup_{i=1}^{k} E_i$ heißen Waldkanten (Baumkanten für $k = 1$).

Bem: Falls G ungerichtet ist:
B/F-Kanten und C-Kanten. //

Beispiel.

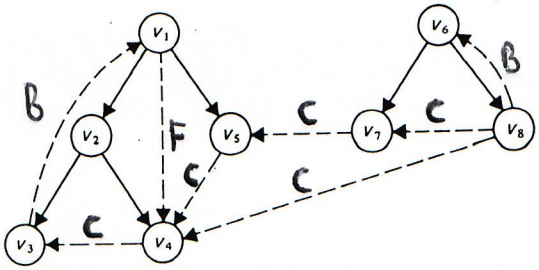
BFS-Wald



keine F-Kanten

Zufall?

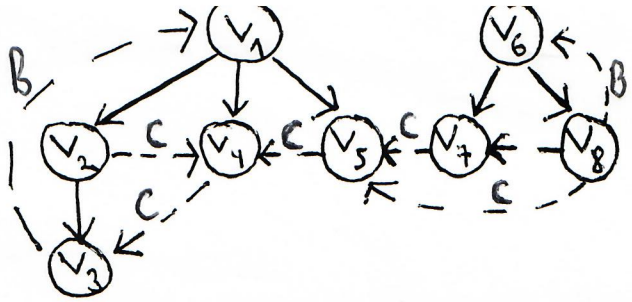
DFS-Wald



C-Kanten laufen alle von rechts nach links.

Zufall?

Sei $T_i = (V_i, E_i)$ ein Baum des BFS-Waldes von G,
$x = root(T_i)$, $y \in V_i$.



(1) $depth_{T_i}(y)$ = Länge des kürzesten Pfades in G von x nach y

Grund: BFS durchläuft die noch nicht besuchten Knoten von x aus "levelweise".

(2) BFS erzeugt niemals F-Kanten.

(3) Eine C-Kante innerhalb T_i kann vom Level j nicht zu einem Level $\geq j+2$ führen.

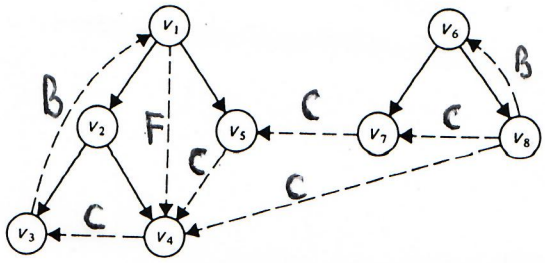
Analog bei ungerichteten Graphen:

(2') Keine F/B-Kanten (nur C-Kanten)

(3') C-Kanten innerhalb eines Baumes nur zwischen aufeinanderfolgenden Levels.

Eigenschaften der DFS:

Illustration



Def.:

x links von y in W

$:\Leftrightarrow$

Die DFS durchläuft
x vor y und $\neg(x \xrightarrow[W]{*} y)$.

(1) DFS erzeugt keine von links nach rechts laufenden C-Kanten.

Grund:

Rückkehr aus einem in x gestarteten rekursiven Aufruf erst, wenn alle noch nicht besuchten direkten Nachfolger von x abgearbeitet sind.

Analog für ungerichtete Graphen:

(1') keine C-Kanten (nur F/B-Kanten)

Folgerung aus (1): Sei x links von y.

Jeder Pfad in G von x nach y enthält mindestens eine B-Kante zu einem gemeinsamen Vorfahren z von x und y.

DFS-Nummern

$n := |V|$.

Wir definieren

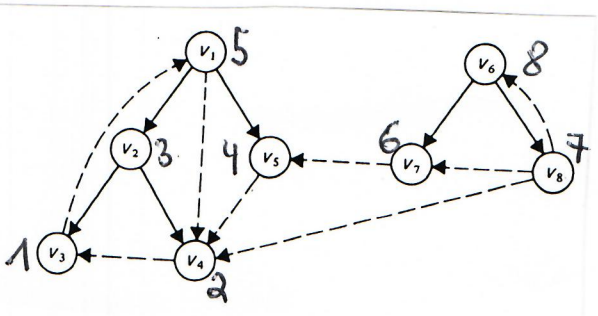
$$N_{\downarrow} : V \longrightarrow \{1, \dots, n\} \qquad \text{(Top-down Nummer)}$$
$$N_{\uparrow} : V \longrightarrow \{1, \dots, n\} \qquad \text{(Bottom-up Nummer)}.$$

$N_{\downarrow}(x)$ verteilt Nummer an x beim Starten des rekursiven Aufrufs auf x,

$N_{\uparrow}(x)$ bei der Rückkehr aus diesem Aufruf.

Aussage: Fortlaufende Verteilung der Nummern $1, \dots, n$.

Illustration



$N_{\downarrow}(v_i) = i$

$N_{\uparrow}$ s. Knotenmarkierung

$N_{\downarrow} \mathrel{\hat{=}}$ Präorder-Numerierung von $T_1, \dots, T_k$

$N_{\uparrow} \mathrel{\hat{=}}$ Postorder-Numerierung von $T_1, \dots, T_k$

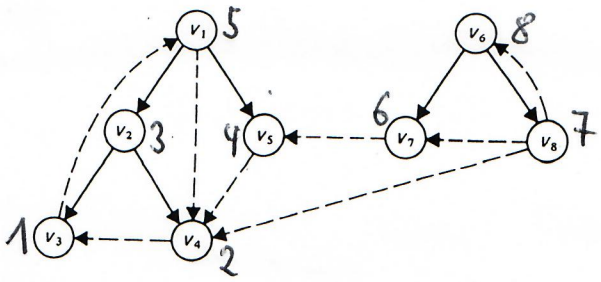
Beobachtungen

Illustration

1) $x \xrightarrow[w]{+} y \iff$

$N_{\downarrow}(x) < N_{\downarrow}(y)$

$N_{\uparrow}(x) > N_{\uparrow}(y)$



2) x links von $y \iff$

$N_{\downarrow}(x) < N_{\downarrow}(y)$, $N_{\uparrow}(x) < N_{\uparrow}(y)$