



Erreichbarkeit

$u \in V$ ist von v aus **erreichbar** wenn es einen Pfad von v nach u gibt.



Operationen

Hier nur **Navigation** in Zeit $\mathcal{O}(1)$ pro Kante:

Gegeben v , finde die ausgehenden Kanten

Zugriff: Auf assoziierte Informationen.

Hier: Knotenmarkierungen.

Lösung: OBdA $V = 1..n$, benutze zusätzliches Feld.

Navigation: Gegeben v , finde die ausgehenden Kanten

(Zeit $\mathcal{O}(1)$ pro Kante)



Graphrepräsentation

Überblick:

- was zählt sind die Basisoperationen
- Adjazenzfelder



Adjazenzfelder

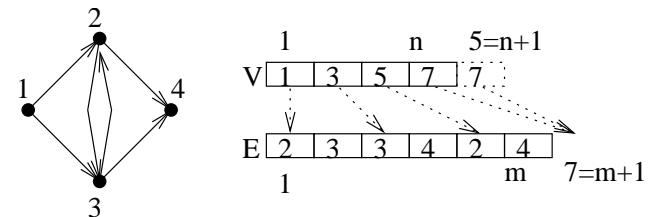
$V = 1..n$

Kantenfeld E speichert **Ziele**

Kanten aus einem Knoten werden gruppiert

Knotenfeld V speichert index der ersten ausgehenden Kanten

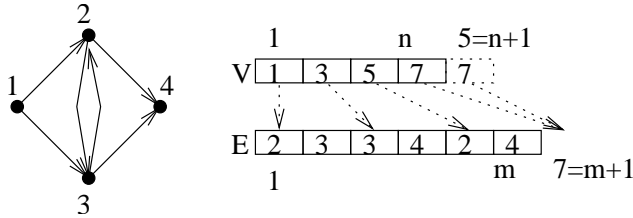
Pseudoeintrag $V[n+1]$ speichert $m+1$



Beispiel: $\text{Ausgangsgrad}(v) = V[v+1] - V[v]$



Adjazenzfelder



- Platz $\mathcal{O}(m+n)$
- Einfach
- Schnelle **Navigation**. $\mathcal{O}(1)$ Zeit.
Cacheeffizienter Zugriff auf Kanten aus einem Knoten.
- Assoziierte Information \rightsquigarrow Records/weitere Felder
- Erweiterbar für weitere Operationen auf **statischen** Graphen

Nur Kanten einfügen etc. ist ein Problem.



Erreichbarkeit mittels **Tiefensuche**

// mark all nodes reachable from s

Procedure reachable(s)

mark s

foreach $(s, v) \in E$ **do**

if v is not marked **then** reachable(v)



Implementierung mittels Adjazenzfeldern

$V[1..n+1]$ // Input

$E[1..m]$ // Graph

mark[$1..n$] = $\langle 0, \dots, 0 \rangle$

Function reachable(s)

 mark[s] := 1

for $e := V[s]$ **to** $V[s+1] - 1$ **do** // for each $e = (s, v) \in E$

$v := E[e]$

if mark[v] = 0 **then** reachable(v)



Korrektheit

folgt unmittelbar aus:

Lemma: reachable markiert genau

$\{v \in V : \exists \text{ Pfad } P = \langle s = v_0, \dots, v = v_k \rangle\}$

Beweis „alles erreichbare wird markiert“:

Induktion über Pfadlänge k

$k = 0$: s wird sofort markiert.

$k \rightsquigarrow k+1$: nach IV wird v_k markiert.

(Am Anfang des Aufrufs reachable(v_k).)

$\xrightarrow{\quad}$ "if mark[$E[v]$] = 0 then reachable(v)" wird

 ausgeführt.

\longrightarrow so oder so wird v_{k+1} markiert



Lemma: reachable markiert genau

$$\{v \in V : \exists \text{ Pfad } P = \langle s = v_0, \dots, v = v_k \rangle\}$$

Beweis „nichts unerreichbares wird markiert“:

Induktion. Übung.



Komplexitätsanalyse

Satz: reachable läuft in Zeit $\mathcal{O}(m+n)$

Beweisskizze:

≤ 1 rekursiver Aufruf pro **Knoten**

≤ 1 Schleifendurchlauf pro **Kante**



Kreis von s erreichbar?

$V[1..n+1]$

// Input

$E[1..m]$

// Graph

$\text{mark}[1..n] = \langle 0, \dots, 0 \rangle$

$\text{onPath}[1..n] = \langle 0, \dots, 0 \rangle$ // flags all nodes on currently explored path

Function reachesCycle(s)

mark[s]:= 1

onPath[s]:= 1

for $e := V[s]$ **to** $V[s+1] - 1$ **do** // foreach $e = (s, v) \in E$

$v := E[e]$

if onPath[v] **then return 1** // “backward edge”

if mark[v] = 0 **then**

if reachesCycle(v) **then return 1**

onPath[s]:= 0 // backtrack

return 0



Korrektheit

Proposition: Die Knoten mit onPath= 1 definieren jeweils einen Pfad

$P = \langle s = v_0, \dots, v_k \rangle$.

→ gemeldete Pfade korrespondieren zu einem über $\langle s, \dots, v_i \rangle$

erreichbaren Kreis $\langle v_i, \dots, v_k, v_i \rangle$.



Korrektheit

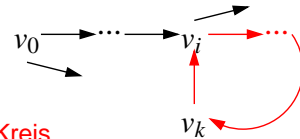
Lemma M: $\text{reachesCycle}(v) = 0 \rightarrow$ alle Knoten werden markiert, die von v über vorher nicht markierte Knoten erreichbar sind.

Beweis: analog Korrektheitsbeweis von reachable.

Lemma: Kein Kreis wird übersehen.

Beweisskizze: Annahme, $\text{reachesCycle}(s) = 0$ aber

\exists Pfad $P = \langle s = v_0, \dots, v_i, \dots, v_k, v_i \rangle$



Alle von s erreichbaren Knoten werden markiert.

OBda Sei v_i der erste markierte Knoten auf dem **Kreis**.

\rightarrow bevor $\text{reachesCycle}(v_i)$ terminiert, werden v_i, \dots, v_k markiert.

\rightarrow beim Aufruf von $\text{reachesCycle}(v_k)$ gilt $\text{onPath}(v_i)$

\rightarrow es wird ein **Kreis** gemeldet !



Gibt es Einen Kreis ?

Initialization for reachesCycle

return $\bigvee_{s \in V} \text{reachesCycle}(s)$

- Markierungen werden nicht zurückgenommen
 \rightsquigarrow Laufzeit $\mathcal{O}(n + m)$
- Korrektheitsbeweis: analog vorher



Komplexitätsanalyse

x **Satz:** reachesCycle läuft in Zeit $\mathcal{O}(m + n)$

Beweis: analog reachable



Starke Zusammenhangskomponenten

$C \subseteq V$ ist **stark zusammenhängend** \Leftrightarrow

$\forall u, v \in C : \exists \text{Pfad } u \rightsquigarrow v.$

$C \subseteq V$ ist **starke Zusammenhangskomponente** \Leftrightarrow

C ist stark zusammenhängend und **maximal**

(d.h. $\nexists w \in V \setminus C : C \cup \{w\}$ ist stark zusammenhängend)



Starke Zusammenhangskomponenten

Satz: Alle starken Zusammenhangskomponenten lassen sich in Zeit $\mathcal{O}(m+n)$ finden.

Beweis: nicht hier.

Idee: **Tiefensuche**.

Invariante: starke Zusammenhangskomponenten bzgl. aller abgearbeiteten Knoten und Kanten sind bekannt.



Topologische Sortierung

Nummeriere die Knoten $V = \langle v_1, \dots, v_n \rangle$, so dass $\forall (v_i, v_j) \in E : i < j$

Das geht genau dann, wenn

$G = (V, E)$ keine gerichteten Kreise enthält —

also G ein **gerichteter aklyischer graph** ist (Directed Acyclic Graph).



Topologische Sortierung

Tiefensuchalgorithmus: (ohne Beweis)

Sequence: order = $\langle \rangle$

foreach $s \in V$ **do**

 explore(s)

Procedure explore(s)

 mark s

foreach $(s, v) \in E$ **do**

if v is not marked **then** explore(v)

 order.pushFront(s)



Topologische Sortierung

Lemma: Jeder DAG enthält einen Knoten mit Ausgangsgrad 0.

Beweis: Angenommen $\forall v \in V : \text{outdegree}(v) > 0$.

$u :=$ any node

$S := \{u\}$

loop

 choose any edge $(u, v) \in E$ // outdegree(u) > 0 !

if $v \in S$ **then** cycle found // Widerspruch (DAG !)

$S := S \cup \{v\}$

$u := v$

Annahme \longrightarrow Endlosschleife $\longrightarrow V$ ist unendlich groß \longrightarrow
Widerspruch.



Topologische Sortierung – noch ein Algorithmus

Starte an beliebigem Knoten.

Suche jeweils

Sequence: order = $\langle \rangle$

while $\exists s \in V : \text{outdegree}(s) = 0$ **do**

 order.pushFront(s)

 remove s and all its incoming edges from G

Übung: Das gleiche Spiel mit Eingangsgrad null.



Wichtiges Muster für Graphenalgorithmien:

- Finde starke Zusammenhangskomponenten
- Starke Zusammenhangskomponenten sind einfach zu verarbeiten
- Kontrahiere starke Zusammenhangskomponenten
- Übrig bleibt ein DAG
- DAGs sind einfach mittels topologischer Sortierung zu bearbeiten

Beispiel transitive Hülle $G = (V, E^+)$, $E^+ := \{(u, v) : \exists \text{path } u \rightsquigarrow v\}$