

Algorithmen und Datenstrukturen

Sommersemester 2020

Stefan Näher

Universität Trier

naeher@uni-trier.de

Vorlesung 14

10. Juni 2020

Topologisches Sortieren: Grundalgorithmus

1. $\text{count} \leftarrow 0$;
2. **while** G besitzt einen Knoten mit $\text{indeg} = 0$ **do**
3. wähle einen solchen Knoten v
4. $\text{ord}[v] \leftarrow ++\text{count}$;
5. streiche v und seine ausgehenden Kante aus G ;
6. **od**
7. **if** $\text{count} \neq n$ **then**
8. ERROR: G enthält einen Kreis
9. **fi**

Problem

Wie finde man in Zeile 2 (schnell) einen Knoten mit Eingangsgrad 0 ?

Ideen

1. Verwalte eine Menge ZERO aller Knoten v mit (aktuell) $\text{indeg}(v) = 0$.
2. Streiche in Zeile 5 nicht wirklich Knoten und Kanten, sondern merke den aktuellen Eingangsgrad jedes Knotens in einem Feld INDEG. Dann braucht man in Zeile 5 nur $\text{INDEG}[w]$ für alle Nachbarknoten w von v um 1 zu vermindern und w in die Menge ZERO aufzunehmen, wenn $\text{INDEG}[w] = 0$.

Topologisches Sortieren: Verfeinerter Algorithmus

// Initialisierung

1. $\text{count} \leftarrow 0$;
2. $\text{ZERO} \leftarrow \emptyset$;
3. **forall** $v \in V$ **do** $\text{INDEG}[v] \leftarrow 0$; **od**
4. **forall** $(v, w) \in E$ **do** $\text{INDEG}[w] ++$; **od**
5. **forall** $v \in V$ **do**
6. **if** $\text{INDEG}[v] = 0$ **then** $\text{ZERO} \leftarrow \text{ZERO} \cup \{v\}$; **fi**
7. **od**

// Hauptschleife

```
8. while ZERO  $\neq \emptyset$  do  
9.   wähle einen beliebigen Knoten  $v \in \text{ZERO}$   
10.  ZERO  $\leftarrow \text{ZERO} \setminus \{v\}$ ;  
11.  ord[v]  $\leftarrow ++\text{count}$ ;  
12.  forall  $w \in V$  mit  $(v, w) \in E$  do  
13.    if  $--\text{INDEG}[w] == 0$  then  
14.      ZERO  $\leftarrow \text{ZERO} \cup \{w\}$ ;  
15.    fi ;  
16.  od  
17. od  
  
18. if count  $\neq n$  then  
19.   ERROR: G ist zyklisch  
20. fi
```

Die Kandidatenmenge ZERO

Operationen auf der Menge ZERO

1. Initialisierung auf leer
2. Einfügen eines Elementes
3. Test auf leer
2. Auswahl + Entfernen eines beliebigen Elements

Mögliche Realisierungen

1. Liste
2. Stack
3. Queue

Alle Operationen haben Laufzeit $O(1)$.

Laufzeitanalyse

Initialisierung

forall $v \in V$ **do** ...

forall $(v, w) \in E$ **do** ...

$\mathcal{O}(n + m)$

Hauptschleife

- Für jedes ausgewählte $v \in \text{ZERO}$ Kosten $\mathcal{O}(1 + \text{outdeg}(v))$
- Jedes v wird maximal einmal ausgewählt.

$$\mathcal{O} \left(\sum_{v \in V} (1 + \text{outdeg}(v)) \right) = \mathcal{O}(n + m)$$

Zusammenfassung

Satz:

Ein gerichteter Graph mit n Knoten und m Kanten kann in Zeit $\mathcal{O}(n + m)$ topologisch sortiert werden.

Korollar:

Es kann in Zeit $\mathcal{O}(n + m)$ getestet werden, ob ein gerichteter Graph azyklisch ist oder nicht.

Zweites grundlegendes Problem auf Graphen: **Systematische Durchmusterung**

Gegeben:

Ein gerichteter Graph $G = (V, E)$ und ein Startknoten $s \in V$.

Problem:

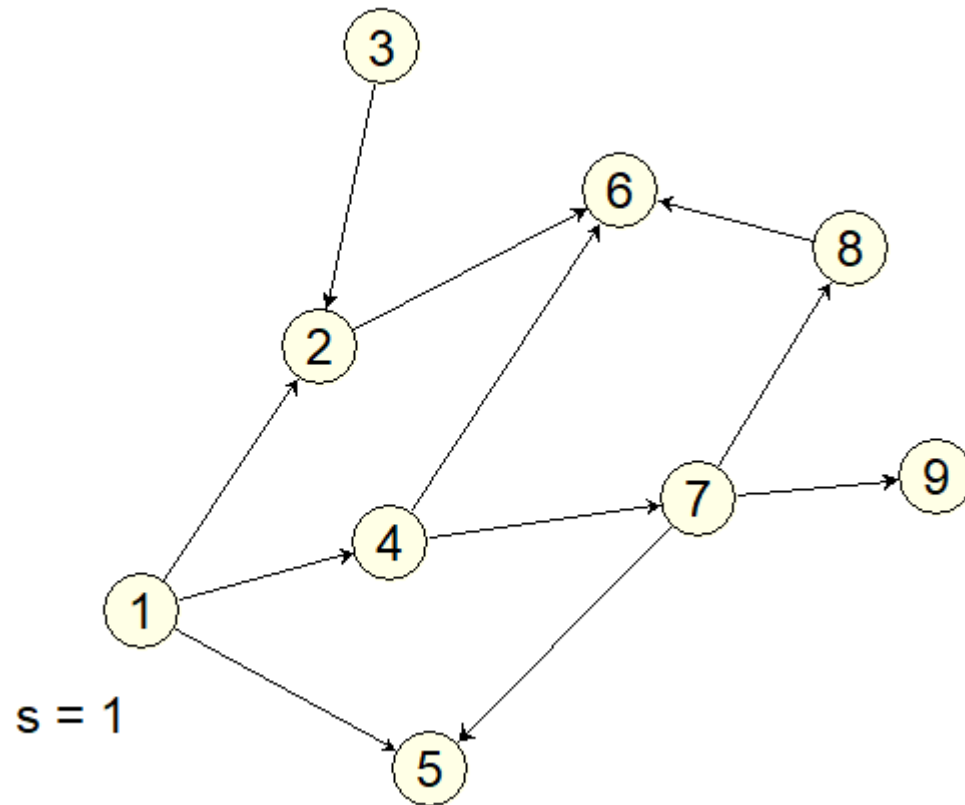
Berechne die Menge S aller Knoten die von s erreichbar sind.

$$S = \{ v \in V \mid \exists \text{ Pfad } P \text{ von } s \text{ nach } v \}$$

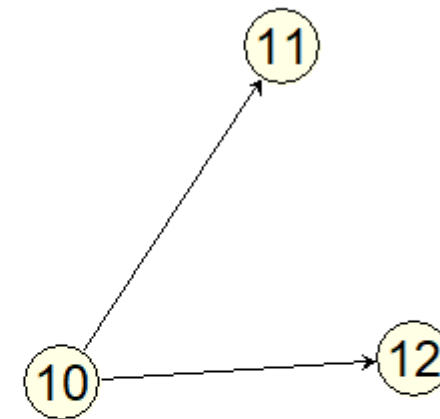
$$\text{oder k\"urzer } S = \{ v \in V \mid s \xrightarrow{*} v \}$$

Zwei grundlegende Strategien: DFS und BFS.

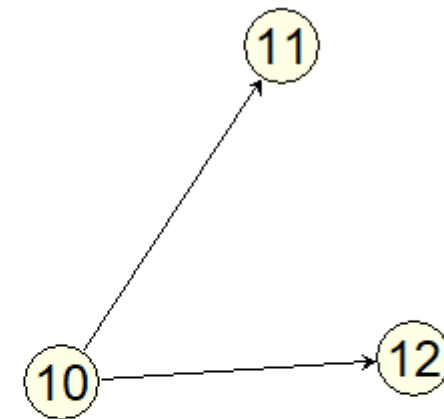
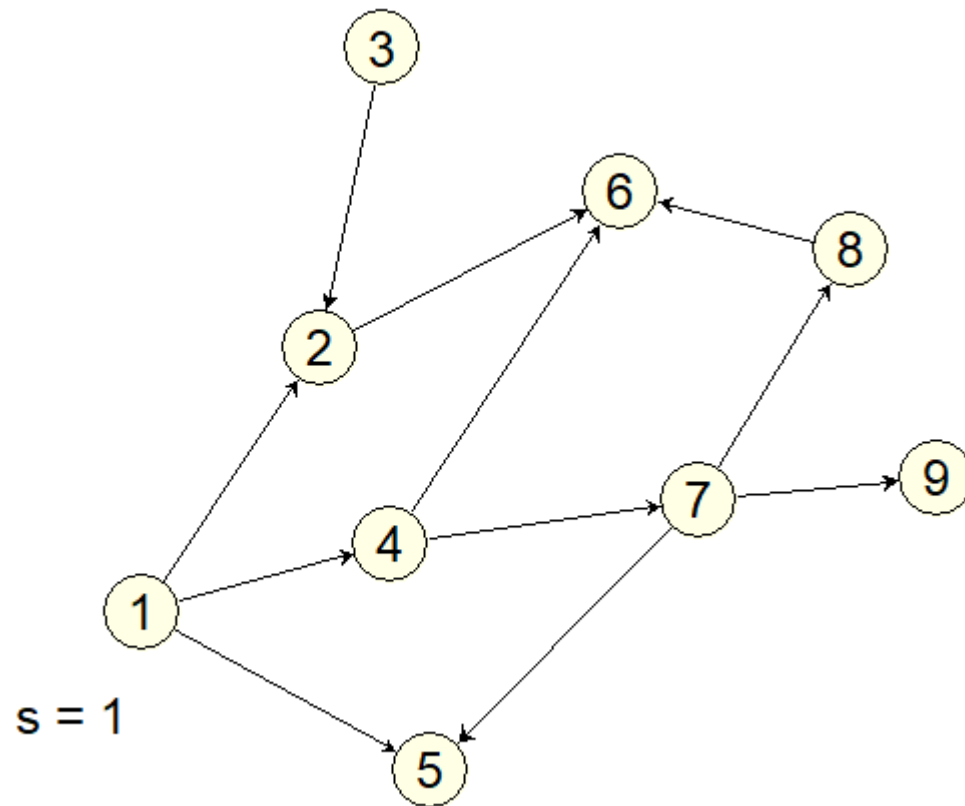
Tiefensuche oder DFS (Depth First Search)



S = 1,2,6,4,7,8,9,5



Breitensuche oder **BFS** (Breadth First Search)



S = 1,2,4,5,6,7,8,9

Grundidee für einen Algorithmus

1. $S \leftarrow \{s\};$
2. markiere alle Kanten $e \in E$ als unbenutzt
3. **while** $\exists$ unbenutzte Kante (v, w) mit $v \in S$ **do**
4. wähle eine solche Kante (v, w)
5. markiere (v, w) als benutzt
6. $S \leftarrow S \cup \{w\};$
7. **od**

Bei Termination enthält S alle von s erreichbaren Knoten.

Noch zu klären:

1. Wie realisiert man die Menge S

Gute Darstellung:

Bitvektor $\text{besucht}[1..n]$ mit $\text{besucht}[v] = \text{true} \iff v \in S$

2. Wie findet man eine unbenutzte Kante aus S heraus ?

Idee: Verwalte die Teilmenge $\tilde{S}$ von Knoten in S , aus denen (noch) unbenutzte Kanten ausgehen können.

Verfeinerter Algorithmus

1. EXPLOREFROM(s)
2. besucht[s] $\leftarrow$ true;
3. $\tilde{S} \leftarrow \{s\}$;
4. **while** $\tilde{S} \neq \emptyset$ **do**
5. $v \leftarrow$ beliebiger Knoten in $\tilde{S}$;
6. $(v, w) \leftarrow$ nächste unbenutzte Kante aus v ;
7. **if** (v, w) existiert nicht
8. **then** $\tilde{S} \leftarrow \tilde{S} \setminus \{v\}$
9. **else if** $\neg$ besucht[w] **then** $\tilde{S} \leftarrow \tilde{S} \cup \{w\}$; **fi**
10. besucht[w] $\leftarrow$ true;
11. markiere (v, w) als benutzt;
12. **fi**
13. **od**

Verwaltung der benutzten Kanten

Verwende einzusätzliches Feld **P[1..n]** von Pointern auf Adjazenzlisten-Elemente

P[v] = Verweis auf die erste unbenutzte Kante in der Adjazenzliste von v.

Siehe Skizze auf nächster Seite.

Initialisierung:

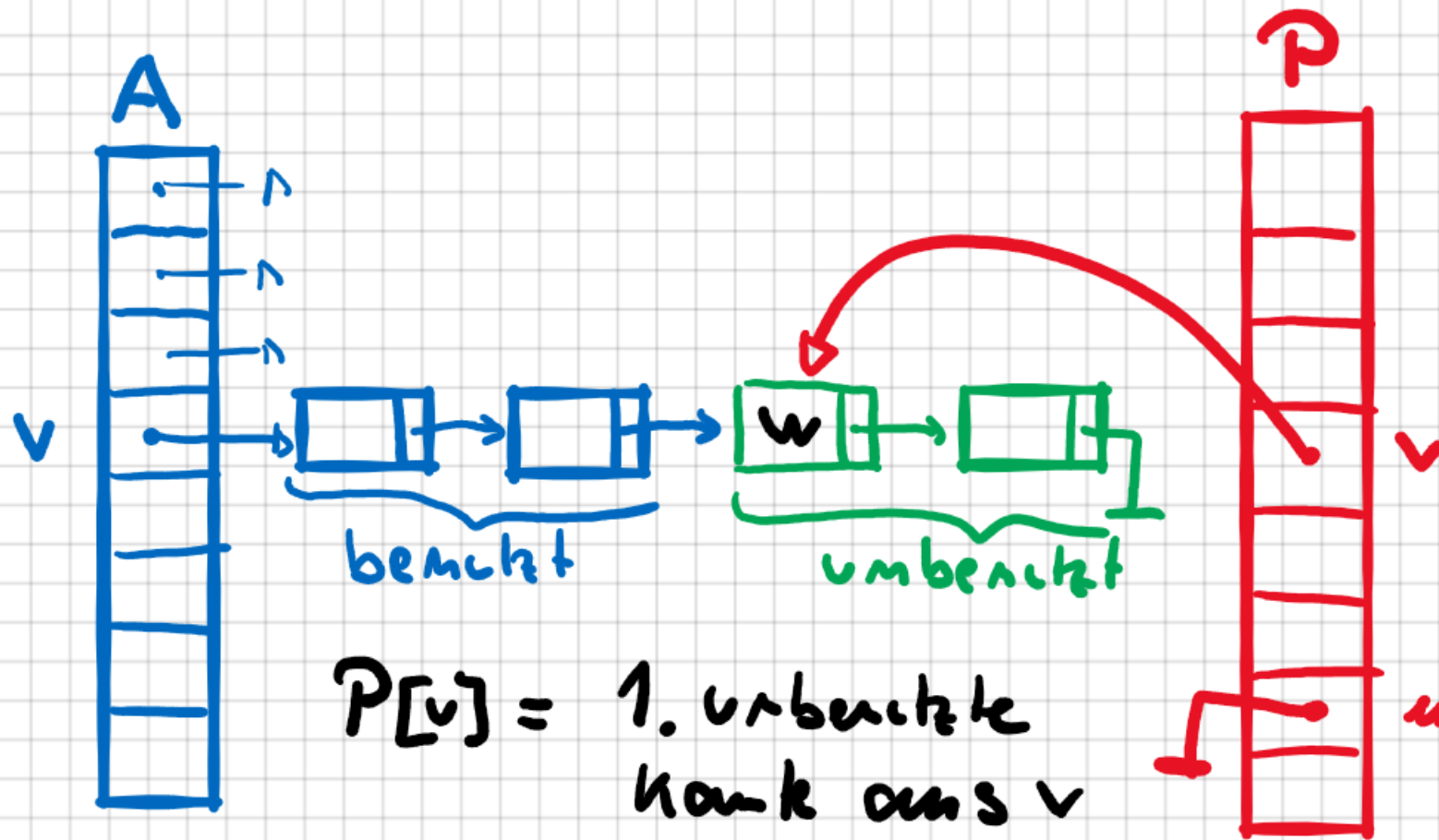
for i = 1 **to** n **do** P[i] $\leftarrow$ A[i] **od**

Test, ob weitere unbenutzte Kante existiert:

if P[v] $\neq$ null **then** ...

Kante als benutzt markieren:

P[v] $\leftarrow$ P[v].next



$P[v] = 1$. unbesuchte
kante aus v

$P[u] = null \rightarrow$ keine unbesuchte kante

2. Verfeinerung

```
1. EXPLOREFROM(s)
2. besucht[s]  $\leftarrow$  true;
3.  $\tilde{S} \leftarrow \{s\}$ ;
4. while  $\tilde{S} \neq \emptyset$  do
5.    $v \leftarrow$  beliebiger Knoten in  $\tilde{S}$ ;
6.   if  $P[v] = \text{null}$ 
7.     then  $\tilde{S} \leftarrow \tilde{S} \setminus \{v\}$ 
8.     else  $w \leftarrow P[v].\text{vertex}$ ;
9.          $P[v] \leftarrow P[v].\text{next}$ ;
10.    if  $\neg \text{besucht}[w]$  then  $\tilde{S} \leftarrow \tilde{S} \cup \{w\}$ ; fi
11.     $\text{besucht}[w] \leftarrow \text{true}$ ;
12.  fi
13. od
```

Die Realisierung von $\tilde{S}$

Durch die Auswahl einer Datenstruktur für die Menge $\tilde{S}$ können verschiedene Durchmusterungs-Strategien realisiert werden.

$\tilde{S}$ als Keller $\longrightarrow$ DFS (Tiefensuche)

$\tilde{S}$ als Schlange $\longrightarrow$ BFS (Breitensuche)

Übungsaufgabe:

Zeigen Sie den Ablauf des Algorithmus auf dem Beispielgraphen

1. mit $\tilde{S}$ als Keller (also DFS)
2. mit $\tilde{S}$ als Schlange (also BFS)

Laufzeitanalyse

Definition: Sei $G = (V, E)$ ein gerichteter Graph und $V' \subseteq V$.
Der Graph $G' = (V', E')$ mit

$$E' = E \cap (V' \times V') = \{ (v, w) \in E \mid v, w \in V' \}$$

heißt **der von V' induzierte Teilgraph von G** .

Lemma:

Sei V_s die beim Aufruf $\text{EXPLOREFROM}(s)$ besuchte Knotenmenge und sei $n_s = |V_s|$ und m_s die Anzahl der Kanten des von V_s induzierten Teilgraphen von G .
Dann hat $\text{EXPLOREFROM}(s)$ die Laufzeit $O(n_s + m_s)$.

Beweis des Lemmas:

Eine Ausführung des Rumpfes der while-Schleife (Zeile 4-13) kostet Zeit $O(1)$. Bei jeder Ausführung wird entweder ein Knoten aus $\tilde{S}$ entfernt (Zeile 7) oder eine Kante benutzt (Zeile 8-9). Da jeder Knoten von V_s genau einmal nach $\tilde{S}$ aufgenommen wird (Test in Zeile 10) und jede Kante, die in einem Knoten von V_s startet, genau einmal benutzt wird (Pointer $P[v]$), kann die while-Schleife höchstens $(n_s + m_s)$ mal ausgeführt werden.

Daraus ergibt sich eine Gesamtlaufzeit von $O(n_s + m_s)$.