Algorithmen & Datenstrukturen

Herbst 2018

Vorlesung 4

Motivation: Irrgarten erkunden (siehe Folien)
z.B. von Punkt A nach B
Kreise vermeiden

z.B. rechte-Hand Strategie (immer rechts – möglichst laufen) funktioniert nicht

Irrgarten ⟷ Graph Knoten: Verzweigungen
Kanten: Verbindungen zwischen diesen

Irrgartenproblem ⟷ Wege finden

Problem: gegeben: Graph und Knoten v
gesucht: Knoten w, die von v erreichbar sind

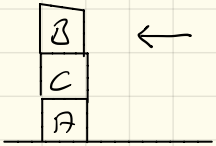
Wir benutzen 2 Datenstrukturen:

Tabelle (Array) der Länge n (n = Anzahl Knoten)

Knoten: A B C D ....
Gesucht?: x x .....

Operationen: – markiere Knoten als gesucht
– teste ob Knoten markiert
beide $O(1)$

Stapel (Stack):



Operationen:
- einen Knoten auf Stapel legen (push)
- auf obersten Knoten zugreifen (top)
- obersten Knoten entfernen (pop)

alle $O(1)$

(wie man das programmiert kommt später)

auf Knoten ≠ obersten kann man nicht zugreifen!

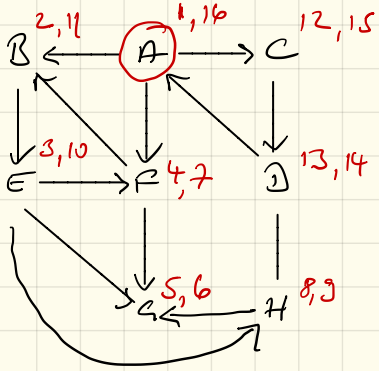
Algorithmus Tiefensuche (depth-first-search, DFS)

Eingabe: (G, v)

1. beginne mit Stapel S der (nur) v enthält
2. wiederhole solange S nicht leer:
   w = oberster Knoten (S)
   falls w besucht, entferne w von Stapel
   falls w nicht besucht:
   markiere w als besucht ("gehe nach w")
   lege alle nicht besuchten
   Nachfolger von w auf Stapel

Iterationen

Beispiel:

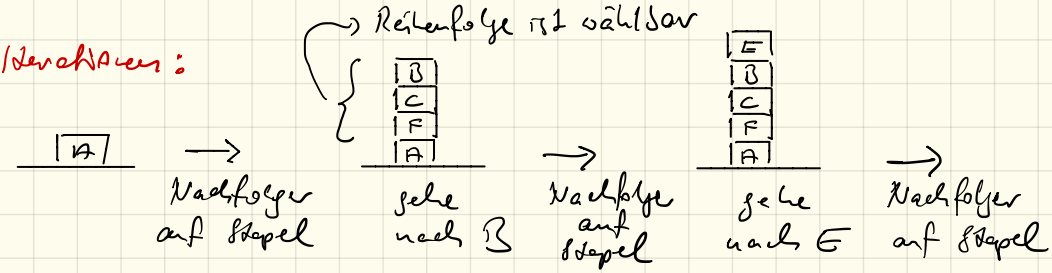


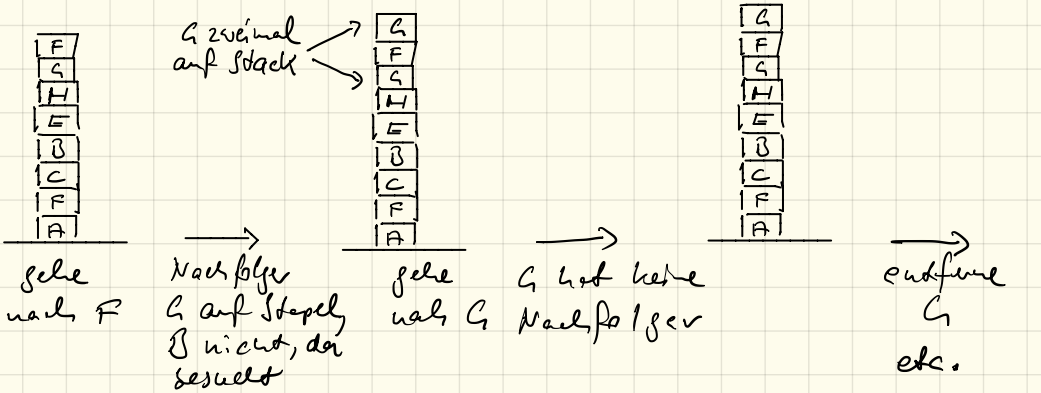
Startknoten: A

A B C D E F G H
x_1 x_2 x_3 x_4 x_5

Iteration des Algos

Betrachten:

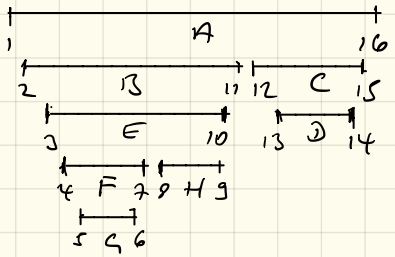




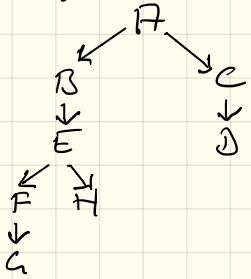
Resultat: alle Knoten sind von A erreichbar

Bedeutung der Knotenzahlen (1–16):

Diese ergeben verschachtelte Intervalle:



Tiefensuchbaum (Teilgraph von G)



Beschreibung des DFS Algorithmus in Pseudocode:

(Pseudocode ist nicht exakt definiert)

1. $S \leftarrow$ Stapel mit Knoten v
2. while S nicht leer do
   a. $w \leftarrow$ top(S)
   b. if w schon besucht then pop(S)
   c. else markiere w als besucht
      push(S, [$w \in N_G^+(w)$ | w nicht besucht])

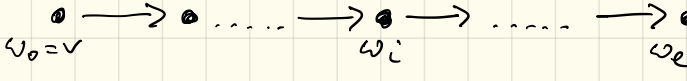
Behauptung: DFS(G, v) besucht genau die Knoten die von v erreichbar sind

Beweis: da wir immer entlang Kanten gehen ist jeder besuchte Knoten erreichbar

(formaler: mache Induktion)

Gegenrichtung: jeder erreichbare Knoten u wird besucht

Betrachte Weg $w_0, \dots, w_\ell$ von $w_0 = v \longrightarrow w_\ell = u$
sei w_i der letzte besuchte Knoten des Weges
(wir wollen zeigen das $i = \ell$)



$\Rightarrow$ w_i wurde in Schritt 2c markiert
$\Rightarrow$ alle Nachfolger gehen auf den Stack
und werden dann auch besucht
$\Rightarrow$ $i < \ell$ ist nicht möglich ∎

Korollar: wenn Knoten $X \subseteq V$ schon markiert sind, dann besucht DFS genau die Knoten die von v erreichbar sind mit Wegen ohne Knoten in X.

Anwendung: Zusammenhangskomponenten (letzte Vorlesung), G ungerichtet

3 Zusammenhangskomponenten

Algorithmus: führe DFS für unbesuchte Knoten aus, solange es unbesuchte gibt

DFS(G):

1. markiere alle $v \in V$ als nicht besucht
2. for $v \in V$
   if v nicht besucht then DFS(v)

Laufzeit: Graph gegeben als Adjazenzliste

$\leq O($Anzahl push/pop/markier Operationen$)$

top Operationen $\leq$ # markier + # pop Operationen

Anzahl

Grund: # top = # Iterationen
und in jeder Iteration gibt es
mindestens ein pop oder markier

also können wir top Ops weglassen
in der Analyse

Beachte: wenn wir die Anzahl der Operationen
ausrechnen, und diese beschränken,
beweisen wir auch dass der Algo
terminiert.

pop Ops = 1 + # push Ops
↑ Aufrufsknoten

der Stapel am Ende leer

markier Ops = n

$V \supseteq W$ = besuchte Knoten während DFS(G, v)

$$\#\text{ push Ops} \leq \sum_{w \in W} \deg_G^+(w)$$

$$\Rightarrow \text{Laufzeit} \leq O\Big(|W| + \sum_{w \in W} \deg_G^+(w)\Big)$$

($|W|$: markier Ops)

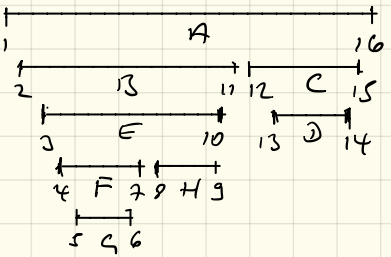
DFS auf ganzem Graphen daher

$$O\Big(|V| + \sum_{v \in V} \deg_G^+(v)\Big) = O(|V| + |E|)$$

Tieferes Verständnis von DFS:

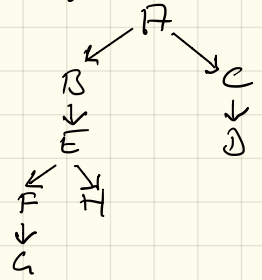
Jedem Knoten v kann man ein Intervall $I_v = [i, j]$, $1 \leq i, j \leq 2n$, zuordnen

Beispiel von zuvor:



→

Tiefensuchbaum (Teilgraph von G)

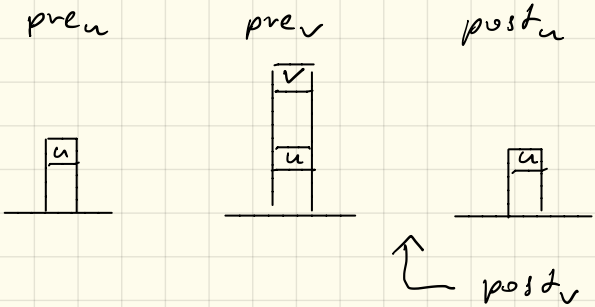


$i = pre_v$: Iteration in der Knoten markiert wird
$i = post_v$: u in der Knoten (zum ersten Mal) vom Stack entfernt ist

Lemma: für alle $u, v \in V$

$I_u \cap I_v = \emptyset$ oder $I_u \subseteq I_v$ oder $I_v \subseteq I_u$

Beweisidee: Wenn $pre_u < pre_v < post_u$
dann ist v über u im Stapel wenn
v besucht wird, d.h. $post_v < post_u$



Kantentypen im Graph: Kante (u,v) ist

vorwärts: $I_u \supseteq I_v$
rückwärts: $I_u \subseteq I_v$
quer: $I_u \cap I_v$

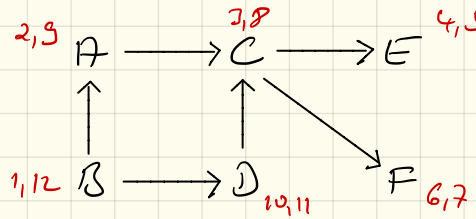
Lemma: für jede Querkante $(u,v) \in E$
liegt I_v vor I_u (= weiter links
im Intervallbild.

Topologische Sortierung durch DFS

Beobachtung: G zyklisch $\Rightarrow$ keine Rückwärtskante bei DFS

(denn jede solche gäbe einen Zyklus)

Theorem: wir erhalten eine topologische Sortierung wenn wir die Knoten nach post absteigend sortieren



Sortierung:

B D A C F E
12 11 9 8 7 5

Beachte: Sortierung nach pre absteigend gilt nicht.

Kürzeste Wege

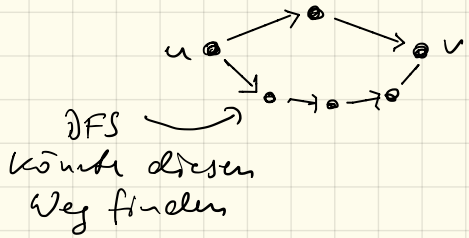
Problem: finde kürzesten Weg von u nach v

Anwendung: Routenplanung (z.B. Google Maps)

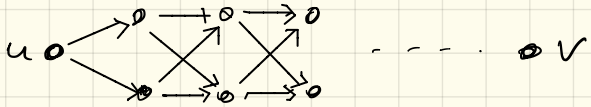
Hier: Länge des Weges = Anzahl Kanten
Abstand $d(u,v)$ = Länge kürzester Weg

Beobachtung: $d(u,v) < \infty \Rightarrow d(u,v) \leq n-1$
("es gibt einen Weg") (da dieser immer ein Pfad ist)

DFS findet Weg, aber nicht immer den kürzesten.



Es kann exponentiell viele Wege geben



(also alle Wege testen ist zu teuer)

Stapel (stack): LIFO = last in, first out
push: oben einfügen
pop: oben entfernen

Schlange (queue): FIFO = first in, first out
enqueue: am Ende einfügen
dequeue: am Anfang entfernen

Breitensuche (breadth first search, BFS): nur ganz kurz