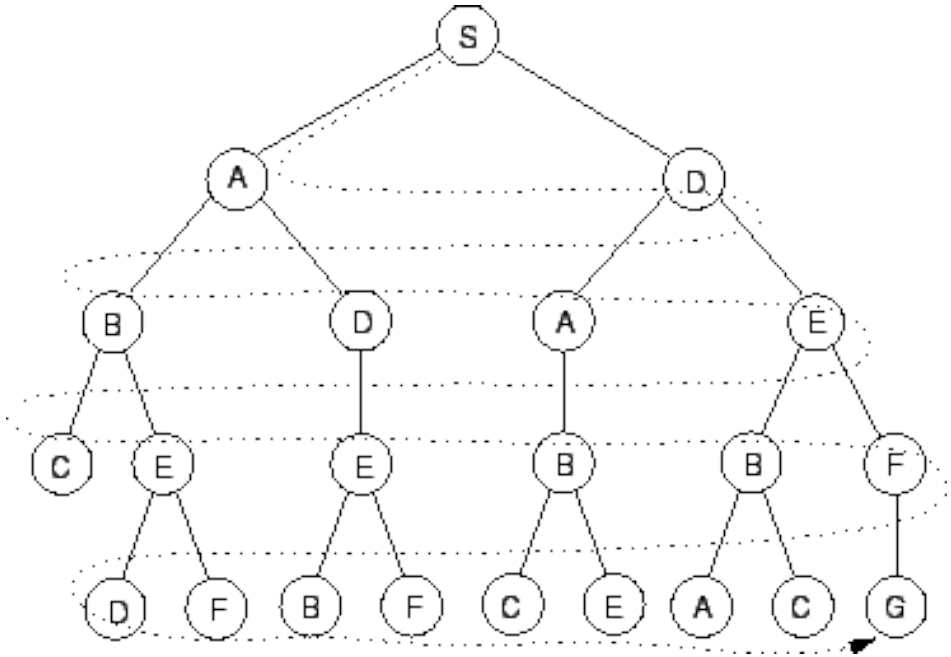
Kapitel 3

Durchsuchen von Graphen



Inhalt

Tiefensuche

- Verallgemeinerung des preorder-Durchlaufprinzips für binäre Bäume
- Kanten werden ausgehend von dem zuletzt entdeckten Knoten v, der mit noch unerforschten Kanten inzident ist, erforscht.
- Erreicht man von v aus einen noch nicht erforschten Knoten w, so verfährt man mit w genauso wie mit v.
- Wenn alle mit w inzidenten Kanten erforscht sind, erfolgt ein Backtracking zu v.
- Die Knoten $v \in V$ erhalten in der Reihenfolge ihres Erreichens eine DFS-Nummer $t(v)$.
  Zu Beginn setzen wir $t(v) = \infty$ für alle $v \in V$.

Algorithmus Tiefensuche

Algorithmus 3.1

Für einen Knoten v sei die Prozedur $DFSEARCH(v)$ wie folgt definiert:

$i := i + 1; t(v) := i; N(v) := \{w | \{v, w\} \in E\};$
while $\exists w \in N(v)$ mit $t(w) = \infty$ *do*
    $N(v) := N(v) \setminus \{w\};$
    $B := B \cup \{\{v, w\}\};$
    $DFSEARCH(w);$
end

Algorithmus 3.2 (Tiefensuche, Depth-First Search)

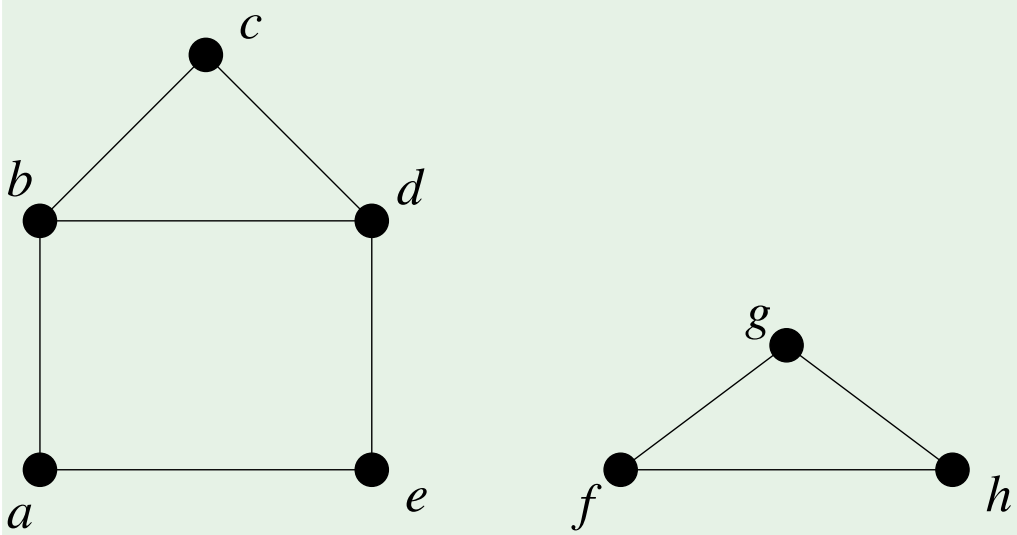
Es sei $G = (V, E)$ ein Graph.

$B := \emptyset; i := 0;$
for all $v \in V$ *do* $t(v) := \infty;$
while $\exists v \in V$ mit $t(v) = \infty$ *do* $DFSEARCH(v);$

Das Ergebnis für $t(v)$ und B hängt i.d.R. von der Repräsenation des Graphen (z.B. als Adjazenzliste) ab.

Beispiel 3.3

Wir betrachten den folgenden Graphen:



Adjazenzlisten:

a	b	$\rightarrow$	e		
b	a	$\rightarrow$	c	$\rightarrow$	d
c	b	$\rightarrow$	d		
d	b	$\rightarrow$	c	$\rightarrow$	e
e	a	$\rightarrow$	d		
f	g	$\rightarrow$	h		
g	f	$\rightarrow$	h		
h	f	$\rightarrow$	g		

Durchlauf und weitere Beispiele: Tafel ✎.

Eigenschaften der Tiefensuche

- Durch DFS wird jeder Knoten besucht und jede Kante wird untersucht.
- Ist G nicht zusammenhängend, dann werden durch DFS nacheinander die Zusammenhangskomponenten ermittelt.
- DFS unterteilt die Kantenmenge E in die Menge der Baumkanten (die Menge B) und die Menge der Rückwärtskanten ($E \setminus B$).
- (V, B) bildet einen aufspannenden Untergraphen von G, der ein Wald ist.
- Ist G zusammenhängend, dann ist (V, B) ein sogenannter aufspannender Baum von G.

Satz 3.4

Der Zeitaufwand für DFS ist $O(|V| + |E|)$.

Definition 3.5

Es sei $G = (V, E)$. Ein Knoten $w \in V$ heißt Nachfolger von $v \in V$ gdw. `DFSEARCH`(w) wird nach Eröffnung und vor Beendigung von `DFSEARCH`(v) aufgerufen.

Lemma 3.6

Wenn $\{v, w\}$ eine Rückwärtskante ist, dann ist v Vorfahr von w oder umgekehrt.

D.h., durch DFS werden keine sogenannten Querkanten konstruiert.

Breitensuche

- Man geht an einem Knoten nicht sofort in die Tiefe, sondern bearbeitet zunächst alle noch nicht erreichten Nachbarn dieses Knotens.
- Anschließend fügt man die Knoten in eine Warteschlange ein.
- Ein Knoten gilt als erreicht, wenn er in die Warteschlange aufgenommen wurde.
- Solange die Warteschlange nicht leer ist, selektiert man einen Knoten aus der Warteschlange und geht wie oben beschrieben vor.
- Die Knoten $v \in V$ erhalten in der Reihenfolge ihres Herausnehmens aus der Warteschlange eine BFS-Nummer $b(v)$.

Algorithmus Breitensuche

Algorithmus 3.7

Für einen Knoten v sei die Prozedur BFSEARCH(v) wie folgt definiert:

$i := i + 1; b(v) := i; N(v) := \{w | \{v, w\} \in E\}$
while $\exists w \in N(v)$ mit $r(w) =$ *false do*
    $N(v) := N(v) \setminus \{w\}$;
    $B := B \cup \{\{v, w\}\}$;
    füge w an W an;
    $r(w) :=$ *true*;
end

Die Prozedur BFSEARCH wird nun für den jeweils ersten Knoten der Warteschlange W ausgeführt.

Algorithmus 3.8 (Breitensuche, Breadth-First Search)

Es sei $G = (V, E)$ ein Graph.

$B := \emptyset; i := 0; W := ();$
for all $v \in V$ *do* $b(v) := \infty; r(v) := false;$ *end*
while $W = ()$ *and* $\exists v \in V : b(v) = \infty$ *do*
    $W := (v);$
    $r(v) := true;$
    while $W \neq ()$ *do*
        wähle ersten Knoten w *aus* W*;*
        entferne w *aus* W*;*
        BFSEARCH(w);
    end
end

Beispiel 3.9

Tafel ✎.

Eigenschaften der Breitensuche

- Jeder Knoten wird besucht und jede Kante wird untersucht.
- Es werden nacheinander die Zusammenhangskomponenten ermittelt.
- (V, B) bildet einen aufspannenden Untergraphen, der ein Wald ist.

Satz 3.10

Der Zeitaufwand für BFS ist $O(|V| + |E|)$.

Halbordnung

Definition 3.11

Eine binäre Relation $R \subseteq V \times V$ heißt Halbordnung auf V gdw. folgendes gilt:

1. $\forall v \in V : (v, v) \in R$ (Reflexivität),
2. $\forall v, w \in V : (v, w) \in R \wedge (w, v) \in R \Rightarrow v = w$ (Antisymmetrie),
3. $\forall u, v, w \in V : (u, v) \in R \wedge (v, w) \in R \Rightarrow (u, w) \in R$ (Transitivität).

Beispiel 3.12

Die Teilbarkeitseigenschaft definiert eine Halbordnung auf den natürlichen Zahlen.

Topologische Ordnung

Lemma 3.13

Es sei $G = (V, A)$ ein DAG. Die Relation $R \subseteq V \times V$ mit

$$(v, w) \in R \iff \exists \text{ gerichteter Weg von } v \text{ nach } w$$

definiert eine Halbordnung auf V.

Definition 3.14

Es sei $G = (V, A)$ ein DAG mit $V = \{v_1, \ldots, v_n\}$. Eine Knotenreihenfolge $L = (v_{i_1}, \ldots, v_{i_n})$ aller Knoten aus V heißt topologische Ordnung von G gdw.

$$(v_{i_j}, v_{i_k}) \in A \Longrightarrow i_j < i_k.$$

Beispiel 3.15

Tafel ✎.

Topologisches Sortieren

Algorithmus 3.16

Gegeben sei ein DAG $G = (V, A)$ beschrieben durch seine Adjazenzlisten. Berechnet wird eine topologische Sortierung, die durch die Numerierung $t(v)$ gegeben ist.

$k := 0; Q := \{v \in V | indeg(v) = 0\}$
while $Q \neq \emptyset$ *do*
    wähle ein v aus Q;
    $k := k + 1;$
    $t(v) := k;$
    $Q := Q \setminus \{v\};$
    for all $(v, w) \in A$ *do*
        $indeg(w) := indeg(w) - 1;$
        if $indeg(w) = 0$ *then* $Q := Q \cup \{w\};$
    end
end

Beispiel 3.17

Tafel ✎.

Satz 3.18

Algorithmus 3.16 liefert in $O(|V| + |E|)$ eine topologische Sortierung $L = (v_{i_1}, \ldots, v_{i_n})$, wobei v_i der Knoten mit $t(v) = i$ ist.

Zusammenfassung des Kapitels

- Tiefensuche: Gehe in die Tiefe solange wie möglich.
- Knotennumerierung und aufspannender, kreisfreier Untergraph
- Breitensuche: Prinzip der Warteschlange
- Anwendungen:
  - systematisches Durchsuchen von Graphen, z.B. Labyrinth
  - Ermittlung von Eigenschaften eines Graphen
- Berechnung von zulässigen Reihenfolgen durch topologische Sortierung der Knoten