5 Graphalgorithmen

5.1 Begriffe und Definitionen

5.1.1 Gerichtete und ungerichtete Graphen

- Ein *Graph* ist ein Paar $G = (V, E)$ mit einer endlichen Menge V von *Knoten* oder *Ecken* (engl. vertices) und einer zugehörigen Menge E von *Kanten* (engl. edges).

- Wenn E eine Teilmenge von $V \times V = \{ (u, v) \mid u, v \in V \}$ ist, heißt der Graph *gerichtet*. (Eine Kante $(u, v) \in E$ geht von u nach v, aber nicht umgekehrt.)

- Wenn E eine Teilmenge von $\{ \{ u, v \} \mid u, v \in V \}$ ist, heißt der Graph *ungerichtet*. (Eine Kante $\{ u, v \} = \{ v, u \} \in E$ geht sowohl von u nach v als auch umgekehrt.)

- Ein ungerichteter Graph kann auch als gerichteter Graph aufgefasst werden, in dem es zu jeder Kante (u, v) auch die entgegengesetzte Kante (v, u) gibt.

- Eine Kante (v, v) oder $\{ v, v \} = \{ v \}$ von einem Knoten v zu sich selbst heißt *Schlinge*.

5.1.2 Adjazenzlisten und -matrizen

- Wenn es in einem Graphen eine Kante von einem Knoten u zu einem Knoten v gibt, heißt u *Vorgänger* von v und v *Nachfolger* von u.

- Die *Adjazenzliste* eines Knotens enthält alle Nachfolger dieses Knotens in irgendeiner Reihenfolge.

- Die *Adjazenzlistendarstellung* eines Graphen besteht aus den Adjazenzlisten aller Knoten des Graphen.

- Die *Adjazenzmatrix* eines Graphen mit $N = |V|$ Knoten ist eine Matrix A mit $N \times N$ Elementen a_{uv}. Jedes a_{uv} ist 1, wenn es eine Kante von u nach v gibt, andernfalls 0.

  Formal: $A: V \times V \to \{0, 1\}$ mit $A(u, v) = \begin{cases} 1, \text{ wenn } (u, v) \in E \text{ bzw. } \{u, v\} \in E \\ 0 \text{ sonst} \end{cases}$

  Folgerung: $|E| \leq |V|^2$

- Die Adjazenzmatrix eines ungerichteten Graphen ist symmetrisch.

- Die Adjazenzlistendarstellung eines Graphen $G = (V, E)$ hat die Größe $O(|V| + |E|)$, die Adjazenzmatrix $O\left(|V|^2\right)$.

5.1.3 Weitere Begriffe

- Ein (einfacher) *Weg* oder *Pfad* von einem Knoten u zu einem Knoten v ist eine Folge paarweise verschiedener Knoten $w_0, \ldots, w_n$ mit $u = w_0$, Kanten von w_{i-1} nach w_i für $i = 1, \ldots, n$ und $w_n = v$.

- Die Anzahl n der Kanten heißt *Länge* des Wegs.
  (Der Fall $n = 0$ als *leerer Weg* von einem Knoten zu sich selbst ist zulässig.)

- Wenn es einen Weg von u nach v gibt, heißt v von u aus *erreichbar*.
  (Insbesondere ist jeder Knoten von sich selbst aus erreichbar.)

- Die *Distanz* $\delta(u, v)$ zwischen u und v ist entweder die Länge eines kürzesten Wegs von u nach v, falls v von u aus erreichbar ist, oder andernfalls ∞.
  (Insbesondere ist $\delta(v, v) = 0$ für jeden Knoten $v \in V$.)

- Wenn $w_0, \ldots, w_n$ ein Weg ist und es eine Kante von w_n nach w_0 gibt, heißt die Knotenfolge $w_0, \ldots, w_n, w_0$ *Zyklus* oder *Kreis*.
  (Beachte: Die Knotenfolge $w_0, \ldots, w_n, w_0$ ist kein Weg, weil ihre Knoten nicht alle verschieden sind.)

- Ein Graph ohne Zyklen heißt *azyklisch* oder *zyklenfrei*.

5.2 Breitensuche (breadth-first search)

5.2.1 Problemstellung

Gegeben

- Graph $G = (V, E)$
- Startknoten $s \in V$

Gesucht

- Alle Knoten $v \in V$, die vom Startknoten s aus erreichbar sind

Genauer

- $\delta(s, v)$ für alle Knoten $v \in V$
- Ein kürzester Weg von s nach v für alle von s aus erreichbaren Knoten $v \in V$

5.2.2 Algorithmus

1 Für jeden Knoten $v \in V \setminus \{ s \}$:

  Setze $\delta(v) = \infty$ und $\pi(v) = \perp$.

2 Setze $\delta(s) = 0$ und $\pi(s) = \perp$.

3 Füge s in eine FIFO-Warteschlange ein.

4 Solange die Warteschlange nicht leer ist:

  1 Entnimm den ersten Knoten u aus der Warteschlange.

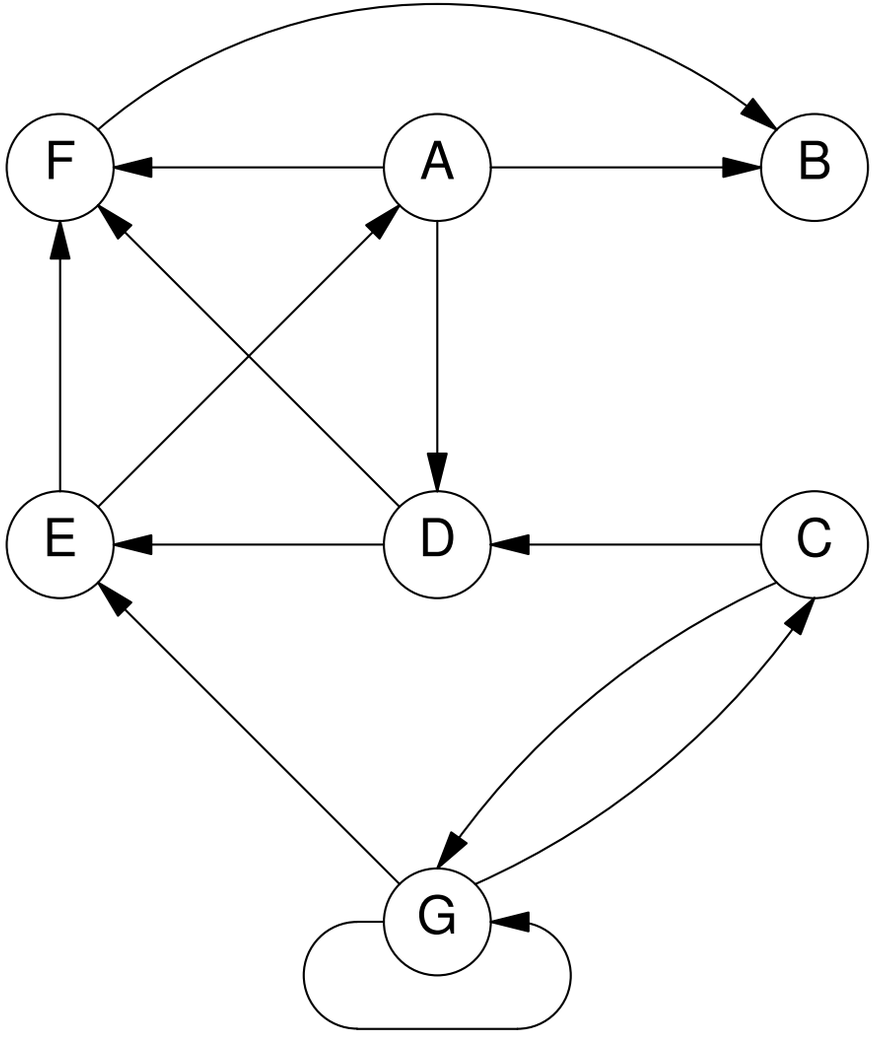
  2 Für jeden Nachfolger v von u:

    Wenn $\delta(v) = \infty$ ist:

    1 Setze $\delta(v) = \delta(u) + 1$ und $\pi(v) = u$.

    2 Füge v am Ende der Warteschlange an.

5.2.3 Beispiel



5.2.4 Laufzeit

- Initialisierung (Schritt 1 und 2): $O(|V|)$
- Eigentliche Suche (Schritt 3 und 4): $O(|V| + |E|)$
  - Die äußere Schleife wird für jeden Knoten höchstens einmal durchlaufen, da jeder Knoten höchstens einmal in die Warteschlange eingefügt wird.
  - Damit wird die innere Schleife insgesamt höchstens $\sum_{u \in V} \gamma(u) = |E|$ mal durchlaufen, wobei $\gamma(u)$ = Grad des Knotens u = Anzahl der Nachfolger von u = Anzahl der von u ausgehenden Kanten.
- Insgesamt also: $O(|V| + |E|)$

5.2.5 Ergebnis

Nach Ausführung des Algorithmus gilt:

- $\delta(v) = \delta(s, v)$ für alle $v \in V$
- Wenn $v \neq s$ von s aus erreichbar ist, dann ist $\pi(v)$ der Vorgänger von v auf einem kürzesten Weg von s nach v, das heißt: Der kürzeste Weg von s nach v lautet $w_n, \ldots, w_0$ mit $w_0 = v$ und $w_i = \pi(w_{i-1})$ für $i = 1, \ldots, n$ mit $n = \delta(v)$.

5.2.6 Vorgängergraph einer Breitensuche (Breitensuchebaum)

- Der Graph $G_\pi = (V_\pi, E_\pi)$ mit
  - $V_\pi = \{ v \in V \mid \delta(v) < \infty \}$ = Menge aller vom Startknoten s aus erreichbaren Knoten
  - $E_\pi = \{ (\pi(v), v) \mid v \in V_\pi \setminus \{ s \} \} = \{ (\pi(v), v) \mid \pi(v) \neq \perp \} \subseteq E$

  ist ein Baum mit Wurzel s,
  der alle von s aus erreichbaren Knoten des Graphen G enthält.

- Auf Ebene d dieses Baums befinden sich alle Knoten v mit $\delta(s, v) = d$.

5.2.7 Anwendungsbeispiel

Automatische Speicherbereinigung (garbage collection), z. B. in Java

- $V = \{ v \mid$ Objekt v wurde mit `new` erzeugt $\}$
- $E = \{ (u, v) \in V \times V \mid$ eine Objektvariable von u verweist auf $v \}$
- $S = \{ s \in V \mid$ eine Klassenvariable oder eine lokale Variable verweist auf Objekt $s \}$
- Die Breitensuche wird (nach einmaliger Initialisierung aller Knoten/Objekte) für alle Objekte $s \in S$ nacheinander ausgeführt.
- $\pi(v)$ wird nicht benötigt.
- Statt $\delta(v)$ genügt die binäre Information, ob v von irgendeinem Objekt $s \in S$ aus erreichbar ist oder nicht.
- Objekte, die auf diese Weise nicht erreichbar sind, können gelöscht werden.

5.3 Tiefensuche (depth-first search) und topologische Sortierung

5.3.1 Problemstellung

- Ordne die Knoten eines Graphen $G = (V, E)$ hierarchisch von links nach rechts an, sodass keine Kanten von links nach rechts zeigen.
- Nebenbei kann überprüft werden, ob der Graph einen Zyklus enthält oder nicht.
- Wenn er azyklisch ist, können die Knoten auch sequentiell von links nach rechts angeordnet werden, sodass alle Kanten von rechts nach links zeigen (*topologische Sortierung*).
- Während der Suche erhält jeder Knoten $v \in V$ einen *Vorgänger* $\pi(v)$ im *Tiefensuchewald* sowie eine *Entdeckungszeit* $\delta(v) \in \mathbb{N}$ und eine *Abschlusszeit* $\varphi(v) \in \mathbb{N}$ mit $1 \leq \delta(v) < \varphi(v) \leq 2\,|V|$.
- Damit besitzt jeder Knoten außerdem zu jedem Zeitpunkt eine gedachte *Farbe*:
  - Ein Knoten ist *weiß*, wenn er noch nicht entdeckt wurde, d. h. wenn er noch keine Entdeckungs- und Abschlusszeit besitzt.
  - Ein Knoten ist *grau*, wenn er gerade bearbeitet wird, d. h. wenn er eine Entdeckungs-, aber noch keine Abschlusszeit besitzt.
  - Ein Knoten ist *schwarz*, wenn seine Bearbeitung abgeschlossen ist, d. h. wenn er sowohl eine Entdeckungs- als auch eine Abschlusszeit besitzt.

5.3.2 Algorithmus

Für jeden Knoten $u \in V$:

Wenn u weiß ist:

1. Setze $\pi(u) = \perp$.
2. Durchsuche den zu u gehörenden Teilgraphen, das heißt:
   1. Setze $\delta(u)$ auf den nächsten Zeitwert aus der Menge $\{1, \ldots, 2 \cdot |V|\}$.
   2. Für jeden Nachfolger v von u:

      Wenn v weiß ist:

      1. Setze $\pi(v) = u$.
      2. Durchsuche rekursiv den zu v gehörenden Teilgraphen.
   3. Setze $\varphi(u)$ auf den nächsten Zeitwert aus der Menge $\{1, \ldots, 2 \cdot |V|\}$.

5.3.3 Beispiel

❒ Siehe § 5.2.3.

5.3.4 Laufzeit

- Die äußere Schleife wird für jeden Knoten genau einmal durchlaufen.
- Die rekursive Operation „Durchsuchen“ wird für jeden Knoten genau einmal ausgeführt.
- Damit wird die innere Schleife in dieser Operation insgesamt $\sum_{u \in V} \gamma(u) = |E|$ mal durchlaufen.
- Damit ergibt sich als Laufzeit: $O(|V| + |E|)$

5.3.5 Vorgängergraph einer Tiefensuche (Tiefensuchewald)

- Der Graph $G_\pi = (V, E_\pi)$ mit $E_\pi = \{ (\pi(v), v) \mid v \in V, \pi(v) \neq \perp \} \subseteq E$ ist eine Menge von Bäumen, d. h. ein Wald.
- Jeder Knoten v mit $\pi(v) = \perp$ ist der Wurzelknoten eines dieser Bäume.

5.3.6 Klassifikation der Kanten

Nach der Ausführung einer Tiefensuche gilt für die Bearbeitungszeiträume $\Sigma(u) = [\delta(u), \varphi(u)]$ und $\Sigma(v) = [\delta(v), \varphi(v)]$ zweier beliebiger Knoten $u, v \in V$ mit $u \neq v$ immer genau eine der folgenden Beziehungen:

- $\Sigma(u) \supset \Sigma(v)$, d. h. $\delta(u) < \delta(v) < \varphi(v) < \varphi(u)$
  - v wurde während der Bearbeitung von u entdeckt.
  - Damit ist v ein direkter oder indirekter Nachfolger von u in einem Baum des Tiefensuchewalds.
  - Wenn es eine Kante von u nach v gibt, geht sie im Tiefensuchewald von oben nach unten und wird deshalb entweder als *Baumkante* (tree edge) – wenn sie zur Menge E_π gehört – oder andernfalls als *Abwärts-* oder *Vorwärtskante* (forward edge) bezeichnet.
- $\Sigma(u) \subset \Sigma(v)$, d. h. $\delta(v) < \delta(u) < \varphi(u) < \varphi(v)$
  - u wurde während der Bearbeitung von v entdeckt.
  - Damit ist v ein direkter oder indirekter Vorgänger von u in einem Baum des Tiefensuchewalds.
  - Wenn es eine Kante von u nach v gibt, geht sie im Tiefensuchewald von unten nach oben und wird deshalb als *Aufwärts-* oder *Rückwärtskante* (back edge) bezeichnet.
  - Eine Schlinge wird ebenfalls so bezeichnet.

- $\Sigma(u) > \Sigma(v)$, d. h. $\delta(v) < \varphi(v) < \delta(u) < \varphi(u)$
  - u wurde nach der Bearbeitung von v entdeckt.
  - Damit ist v weder ein Nachfolger noch ein Vorgänger von u in einem Baum des Tiefensuchewalds.
  - Wenn es eine Kante von u nach v gibt, geht sie im Tiefensuchewald von rechts nach links und wird deshalb als *Querkante* (cross edge) bezeichnet.
- $\Sigma(u) < \Sigma(v)$, d. h. $\delta(u) < \varphi(u) < \delta(v) < \varphi(v)$
  - v wurde nach der Bearbeitung von u entdeckt.
  - Damit ist v weder ein Nachfolger noch ein Vorgänger von u in einem Baum des Tiefensuchewalds.
  - Wenn es eine Kante von u nach v gäbe, müsste sie im Tiefensuchewald von links nach rechts gehen.
  - Aber wenn es eine solche Kante gäbe, dann hätte der Algorithmus den Knoten v spätestens während der Bearbeitung von u über diese Kante entdeckt, d. h. dann würde entweder $\Sigma(v) < \Sigma(u)$ (Entdeckung und Bearbeitung von v bereits vor der Bearbeitung von u) oder $\Sigma(v) \subset \Sigma(u)$ (Entdeckung und Bearbeitung von v während der Bearbeitung von u) gelten.
  - Daher gibt es im Tiefensuchewald niemals Kanten von links nach rechts.

Aufgrund der Rekursionsstruktur des Algorithmus ist eine Überlappung der Bearbeitungszeiträume, d. h. $\delta(u) < \delta(v) < \varphi(u) < \varphi(v)$ oder $\delta(v) < \delta(u) < \varphi(v) < \varphi(u)$, nicht möglich.

5.3.7 Topologische Sortierung

Problemstellung

- Die Kanten eines azyklischen gerichteten Graphen können als Abhängigkeitsbeziehungen zwischen den Knoten interpretiert werden, d. h. $(u, v) \in E$ bedeutet, dass Knoten u irgendwie von Knoten v abhängt.
- Gesucht ist eine sequentielle Anordnung der Knoten von links nach rechts, in der jeder Knoten erst nach den Knoten kommt, von denen er abhängt, d. h. in der alle Kanten von rechts nach links verlaufen.

Lösung: Erweiterte Tiefensuche

- Für jeden Nachfolger v von u wird in Schritt 2.2 der Tiefensuche zusätzlich überprüft, ob er grau ist. Wenn ja, handelt es sich bei der Kante (u, v) um eine Rückwärtskante. In diesem Fall enthält der Graph einen Zyklus und kann daher nicht topologisch sortiert werden.
- Wenn jeder Knoten u nach Abschluss seiner Verarbeitung am Ende einer linearen Liste angefügt wird, enthält diese Liste nach Beendigung der Tiefensuche die Knoten in der gewünschten Reihenfolge.

Anwendungsbeispiele

- Abhängigkeiten zwischen Software-Komponenten, Begriffsdefinitionen, ...

5.4 Zusammenhangskomponenten

5.4.1 Definitionen

- Eine *Verbindung* zweier Knoten u und v ist eine Folge paarweise verschiedener Knoten $w_0, \ldots, w_n$ mit $u = w_0$, Kanten von w_{i-1} nach w_i oder umgekehrt für $i = 1, \ldots, n$ und $w_n = v$. (In einem ungerichteten Graphen sind Verbindung und Weg gleichbedeutend.)
- Ein Graph heißt *zusammenhängend*, wenn es von jedem Knoten eine Verbindung zu jedem anderen Knoten gibt.
- Eine *Zusammenhangskomponente* eines Graphen ist eine Äquivalenzklasse der Relation „es gibt eine Verbindung von u nach v“, d. h. zwei Knoten gehören genau dann zur gleichen Zusammenhangskomponente, wenn es eine Verbindung zwischen ihnen gibt.
- Ein gerichteter Graph heißt *stark zusammenhängend*, wenn es von jedem Knoten einen Weg zu jedem anderen Knoten gibt, d. h. wenn jeder Knoten von jedem anderen Knoten aus erreichbar ist.
- Eine *starke Zusammenhangskomponente* eines gerichteten Graphen ist eine Äquivalenzklasse der Relation „u ist von v aus erreichbar und umgekehrt“, d. h. zwei Knoten u und v gehören genau dann zur gleichen starken Zusammenhangskomponente, wenn u von v aus erreichbar ist und umgekehrt.

- Ein Graph ist genau dann (stark) zusammenhängend, wenn er genau eine (starke) Zusammenhangskomponente besitzt.
- Für einen gerichteten Graphen $G = (V, E)$ heißt der Graph $G^T = \left(V, E^T\right)$ mit $E^T = \{ (v, u) \mid (u, v) \in E \}$ *transponierter Graph* von G.
  (Die Adjazenzmatrix von G^T ist die transponierte Matrix der Adjazenzmatrix von G.)
- Offensichtlich besitzen G und G^T die gleichen starken Zusammenhangskomponenten.

5.4.2 Bestimmung von Zusammenhangskomponenten

- Die Zusammenhangskomponenten eines ungerichteten Graphen können direkt durch eine Tiefensuche bestimmt werden:
  Jeder Baum des resultierenden Tiefensuchewalds entspricht direkt einer Zusammenhangskomponente.

- Die Zusammenhangskomponenten eines gerichteten Graphen $G = (V, E)$ können analog durch eine Tiefensuche im Graphen $G' = \left(V, E \cup E^{\mathrm{T}}\right)$ bestimmt werden.

- Die starken Zusammenhangskomponenten eines gerichteten Graphen $G = (V, E)$ können wie folgt durch zwei aufeinanderfolgende Tiefensuchen bestimmt werden:
  - Führe eine erste Tiefensuche auf G aus, um die Abschlusszeiten $\varphi(u)$ aller Knoten $u \in V$ zu bestimmen.
  - Führe eine zweite Tiefensuche auf G^{T} aus, in der die Knoten u in der äußeren Schleife in absteigender Reihenfolge dieser Abschlusszeiten $\varphi(u)$ durchlaufen werden.
  - Jeder Baum des resultierenden Tiefensuchewalds der zweiten Tiefensuche entspricht einer starken Zusammenhangskomponente von G (und von G^{T}).
  - Beispiel: Siehe § 5.2.3

5.4.3 Korrektheit der Bestimmung starker Zusammenhangskomponenten

Behauptung

Jeder Baum des zweiten Tiefensuchewalds ist eine starke Zusammenhangskomponente des Graphen G.

Beweis durch vollständige Induktion

nach der Anzahl n der bis jetzt von der zweiten Tiefensuche ermittelten Bäume

Induktionsanfang $n = 0$: Hier ist nichts zu zeigen.

Induktionsschritt $n \to n + 1$:

- Nach Induktionsvoraussetzung sind die bis jetzt ermittelten n Bäume starke Zusammenhangskomponenten des Graphen.
- Zu zeigen:
  Der nächste ermittelte Baum ist ebenfalls eine starke Zusammenhangskomponente.
- Sei r der Knoten, für den in der äußeren Schleife der zweiten Tiefensuche als nächstes die Operation „Durchsuchen“ ausgeführt wird und der somit die Wurzel dieses nächsten Tiefensuchebaums wird.

- Die zweite Tiefensuche findet ausgehend von diesem Wurzelknoten r
  - keine Knoten, die im ersten Tiefensuchewald links von r liegen, weil es in diesem Tiefensuchewald gemäß § 5.3.6 keine Kanten von links nach rechts und dementsprechend nach Transponierung des Graphen keine Kanten von rechts nach links gibt;
  - keine Knoten, die im ersten Tiefensuchewald rechts oder oberhalb von r liegen, weil diese eine größere Abschlusszeit als r besitzen und deshalb von der zweiten Tiefensuche (deren Hauptschleife nach absteigenden Abschlusszeiten durchlaufen wird) bereits früher gefunden wurden;
  - folglich nur Knoten, die im ersten Tiefensuchewald unterhalb von r liegen und deshalb auch in G von r aus erreichbar sind und somit zur gleichen starken Zusammenhangskomponente wie r gehören.
- Außerdem findet eine Tiefensuche immer *alle* erreichbaren Knoten, die nicht schon früher gefunden wurden.
  Das bedeutet umgekehrt für alle Knoten v, die dabei nicht gefunden werden:
  - Entweder ist v in G^T nicht von r aus erreichbar und gehört deshalb nicht zur starken Zusammenhangskomponente von r.
  - Oder v wurde bereits früher gefunden und gehört deshalb nach Induktionsvoraussetzung zu einer anderen starken Zusammenhangskomponente.
- Also werden *genau* diejenigen Knoten gefunden, die zur starken Zusammenhangskomponente von r gehören.