Albert-Ludwigs-Universität
Institut für Informatik
Prof. Dr. Fabian Kuhn
P. Bamberger, Y. Maus, P. Schneider

Informatik 2 - Sommersemester 2018
Musterlösung Übungsblatt 7

Abgabe: Montag, 18. Juni, 14:00 Uhr

Aufgabe 1: Tiefensuche in ungerichteten Graphen *(7 Punkte)*

Sei $G = (V, E)$ ein Graph und $(u, v) \in E$ eine Kante. Wird während der Ausführung von `DFS-visit`(u) ein rekursiver Aufruf `DFS-visit`(v) gestartet, dann klassifizieren wir (u, v) wie folgt:

- v ist weiß $\Longrightarrow$ (u, v) ist eine Baumkante.
- v ist grau $\Longrightarrow$ (u, v) ist eine Rückwärtskante.
- v ist schwarz und v wurde *nach* u grau $\Longrightarrow$ (u, v) ist eine Vorwärtskante.
- v ist schwarz und v wurde *vor* u grau $\Longrightarrow$ (u, v) ist eine Querkante.

Zeigen Sie dass `DFS-visit`(s) in einem *ungerichteten* Graphen $G = (V, E)$ und $s \in V$ nur Baumkanten und Rückwärtskanten erzeugt.

Musterlösung

Wir zeigen, dass bei einem Aufruf von `DFS-visit`(v) während der Ausführung von `DFS-visit`(u) der Knoten v nicht schwarz sein kann. Zwecks eines Widerspruchsbeweises nehmen wir aber genau das an. Sei w der Nachbar-Knoten von v über den v entdeckt wurde. D.h. `DFS-visit`(v) wurde das erste mal in `DFS-visit`(w) aufgerufen, weshalb $\{w, v\}$ Baumkante ist. Insbesondere ist $u \neq w$ da v in der genannten Situation sonst weiß wäre.
Da v schon schwarz ist wurden zuvor bereits alle anderen Nachbarn $x \neq w$ von v fertig abgearbeitet mittels eines rekursiven Aufrufes `DFS-visit`(x) (vgl. Pseudocode aus der Vorlesung). D.h. alle Nachbarn $x \neq w$ von v sind schon schwarz. Aber u ist ebenfalls Nachbar von v, ist aber noch grau, ein Widerspruch.

Aufgabe 2: Graphen auf Kreisfreiheit testen *(6 Punkte)*

Sei $G = (V, E)$ ein ungerichteter Graph repräsentiert durch eine Adjazenzliste. Beschreiben Sie einen Algorithmus oder geben Sie Pseudocode an, mit dem man G in $\mathcal{O}(|V|)$ Schritten auf Kreisfreiheit testen kann.

Hinweis: Welche obere Schranke für die Anzahl der Kanten hat ein ungerichteter kreisfreier Graph?

Musterlösung

Variante 1: Im ersten Schritt iterieren wir die Adjazenzliste und zählen Kanten (genauer: wir zählen die Liste der Nachbarn eines Knotens und gehen zur Liste des nächsten Knoten wenn wir die Nachbarn des vorigen Knotens gezählt haben). Wenn wir bei $2|V|$ angekommen sind, wissen wir, dass G mindestens $|V|$ Kanten hat (jede Kante eines ungerichteten Graphen kommt in der Adjazenzliste höchstens

zwei mal doppelt vor). Sollte dies der Fall sein, dann hat G einen Kreis enthält (ein ungerichteter kreisfreier Graph hat maximal $|V| - 1$ Kanten).

Wurden weniger als $2|V|$ Kanten gezählt, startet man vom ersten Knoten der Liste ausgehend eine Breitensuche. Trifft man dabei auf einen bereits markierten (nicht weißen) Knoten, hat man einen Kreis gefunden (dies gilt nicht in gerichteten Graphen!). Hat man alle Knoten in dieser Zusammenhangskomponente besucht und keinen Kreis gefunden, startet man eine neue Breitensuche ausgehend vom ersten Knoten der Liste, der noch nicht besucht wurde. Dieses Verfahren wird wiederholt, bis entweder ein Kreis gefunden wurde oder alle Knoten besucht wurden. Dies dauert $\mathcal{O}(|V| + |E|)$. Da man durch den ersten Schritt $|E| < 2|V|$ sichergestellt hat, beträgt die Laufzeit $\mathcal{O}(|V|)$.

Variante 2: Es reicht auch eine Breiten- oder Tiefensuche auszuführen, bei der man abbricht sobald man einen Kreis gefunden hat. Dies ist der Fall wenn man auf einen Knoten trifft der nicht mehr weiß gefärbt ist. Da ein kreisfreier Graph höchstens $|V| - 1$ Kanten haben kann müssen wir maximal $|V|$ Kanten untersuchen bis feststeht, dass der Graph einen Kreis hat, was wir entsprechend zurückgeben. Falls die Breiten-/Tiefensuche (jeweils neu gestartet auf noch unbesuchte Zusammenhangskomponenten, s.o.) den kompletten Graph traversiert ohne einen Kreis zu finden geben wir zurück dass der Graph kreisfrei ist. Das bedeutet, dass die Breiten-/Tiefensuche zusammen mit dieser Abbruchbedingung höchstens $\mathcal{O}(|V|)$ Zeitschritte benötigt um einen Kreis zu finden.

Aufgabe 3: Minimaler Spannbaum *(7 Punkte)*

Gegeben sei ein gewichteter, ungerichteter, zusammenhängender Graph G. Beweisen Sie, dass folgender Algorithmus terminiert und einen minimalen Spannbaum von G berechnet:

Algorithm 1: `MST`$\big(G = (V, E, w)\big)$

while G *has cycle* $C \subseteq E$ **do**
  Let $e \in C$ with maximum weight.
  $E \leftarrow E \setminus \{e\}$
return G

Hinweise: Zur Terminierung: Sie dürfen davon ausgehen, dass das Ermitteln eines Zyklus in G in endlich vielen Schritten möglich ist. Zur Korrektheit: Beweisen Sie die Invariante, dass der Restgraph $(V, E\setminus\{e\})$ nach Entfernen der schwersten Kante $e \in C$ noch einen MST vom Ursprungsgraphen (V, E) enthält.

Musterlösung

Beweis dass der Algorithmus terminiert: Solange G einen Kreis hat entfernen wir eine Kante, was bedeutet dass wir nach höchstens $\mathcal{O}(|E|)$ Schritten fertig sind. *(2 Punkte)*

Vorbedingung: Zu Beginn enthält G einen MST, da G zusammenhängend ist.

Invariante: Sei C ein Zyklus und $e = \{u, v\} \in C$ die schwerste Kante, welcher der Algorithmus löscht. Sei $T = (V, E_T)$ ein MST, welcher vor der Löschung von e in G enthalten war. Falls $e \notin E_T$, ist T auch nach der Löschung in G enthalten. Falls $e \in E_T$, definiere $T_1 := (V, E_{T_1})$ mit $E_{T_1} = E_T\setminus\{e\}$. T_1 besteht aus zwei Zusammenhangskomponenten, eine davon enthält u und die andere v. $C\setminus\{e\}$ ist ein Pfad (im ursprünglichen Graphen), welcher u und v verbindet. Folglich muss es eine Kante $e' \in C\setminus\{e\}$ geben, welche beide Zusammenhangskomponenten verbindet. Also ist $T_2 := (V, E_{T_2})$ mit $E_{T_2} = E_{T_1} \cup \{e'\}$ ein Spannbaum und wegen $w(e) \geqslant w(e')$ hat T_2 kein größeres Gewicht als T. Also ist T_2 auch ein MST, welcher in G (nach Löschung von e) enthalten ist.

Nachbedingung: Da man alle Kreise eliminiert, erhält man einen kreisfreien Graphen. Dieser muss nach der obigen Invariante auch ein MST sein. *(5 Punkte)*