Karlsruher Institut für Technologie

Institut für Theoretische Informatik
T.T.-Prof. Dr. Thomas Bläsius
https://scale.iti.kit.edu

Übungsblatt 10

Algorithmen I – Sommersemester 2022

Abgabe im ILIAS bis 06.07.2022, 14:00 Uhr

Bitte beschrifte Deine Abgabe gut sichtbar mit Deinem Namen und Deiner Matrikelnummer. Achte insbesondere bei handschriftlichen Abgaben auf Lesbarkeit und genügend Platz für Korrektur-Anmerkungen. Die Abgabe erfolgt über das Übungsmodul in der Gruppe Deines Tutoriums im ILIAS. Gib Deine Ausarbeitungen in *einer* PDF-Datei ab. Achte darauf, effiziente Algorithmen zu formulieren, also solche mit möglichst geringer asymptotischer Laufzeit!
Wenn du die Korrektheit eines Algorithmus begründen oder dessen Laufzeit analysieren sollst, tue dies getrennt von der Beschreibung des Algorithmus.
Wenn nicht anders spezifiziert oder aus dem Kontext ersichtlich, bezeichnen wir mit Graph einen einfachen ungerichteten Graphen.

Aufgabe 1 - Lauter Bäume (4 Punkte)

Entscheide für jede der folgenden Aussagen, ob sie wahr oder falsch ist und begründe deine Antwort.

1. Sei $G = (V, E)$ ein gerichteter Graph und seien $u, v \in V$. Angenommen, es gibt einen Pfad von u nach v in G. Dann liegt v in jedem an s gewurzeltem DFS-Baum von G im Teilbaum unter u. (1 Punkt)

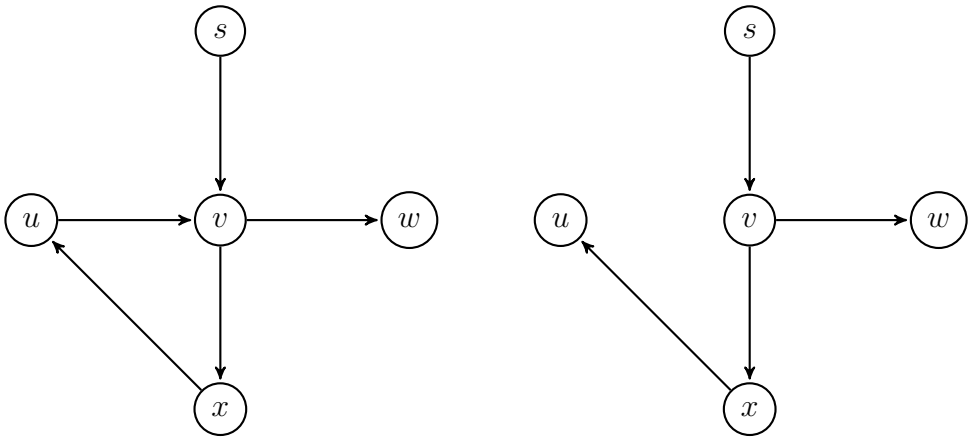
2. Sei $G = (V, E)$ ein gerichteter Graph und seien $u, v \in V$. Angenommen, es gibt einen Pfad von u nach v in G. Dann wird v von einer Tiefensuche auf G entdeckt, bevor u fertig abgearbeitet wurde. (1 Punkt)

3. Wir nennen einen gerichteten Graphen $G = (V, E)$ *semi-zusammenhängend*, wenn für alle Paare von Knoten $u, v \in V$ gilt, dass u von v aus erreichbar ist oder v von u aus (oder beides). Es gibt gerichtete Graphen mit einer eindeutigen Quelle, die keine Kreise enthalten und *nicht* semi-zusammenhängend sind. (1 Punkt)
Hinweis: Eine Quelle ist ein Knoten ohne eingehende Kanten.

4. Sei $G = (V, E)$ ein ungerichteter Graph und $\{v, w\} \in E$ eine Kante von G. Dann ist v in jedem DFS-Baum von G Nachfahre oder Vorfahre von w. (1 Punkt)

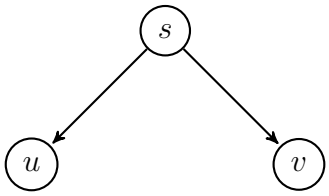
Lösung 1

1. **Falsch**. Ein Gegenbeispiel ist der linke Graph, zu welchem der rechte Graph ein DFS-Baum ist:



2. **Falsch**. Betrachte den Graphen aus der vorherigen Teilaufgabe. Wird bei einer Tiefensuche von s aus der Knoten x vor w exploriert, so ist u von der Tiefensuche abgearbeitet, bevor der Knoten w entdeckt wird, obwohl es einen Pfad von u zu w gibt.

3. **Wahr**. Ein Beispiel ist der folgende Graph:



4. **Wahr**. Angenommen, dem wäre nicht so. Dann liegen v und w in unterschiedlichen Teilbäumen eines DFS-Baumes. Dann wäre $\{v, w\}$ weder Baum- noch Rückkante. Da G ungerichtet ist, ist dies nicht möglich.

Aufgabe 2 - Cut-Vertices (9 Punkte)

In einem Graph $G = (V, E)$ bezeichnen wir einen Knoten $v \in V$ als *cut-vertex*, wenn G durch das Löschen von v in mehrere Zusammenhangskomponenten zerfällt. In dieser Aufgabe überlegen wir uns, wie alle cut-vertices eines Graphen mittels DFS gefunden werden können.

1. Zeige, dass die Wurzel eines DFS-Baums genau dann ein cut-vertex von G ist, wenn sie mehr als ein Kind hat. (3 Punkte)

2. Zeige, dass ein Knoten v, der nicht die Wurzel des DFS-Baums ist, genau dann ein cut-vertex von G ist, wenn gilt: v hat ein Kind u im DFS-Baum, sodass es keine Rückkante aus dem Teilbaum unter u zu Vorfahren von v gibt. (2 Punkte)

3. Wie kann anhand der low-Werte im DFS-Baum entschieden werden, ob ein Knoten v, der nicht die Wurzel des DFS-Baums ist, ein cut-vertex ist? Begründe deine Antwort. (1 Punkt)

4. Gib einen Algorithmus an, der in $O(n + m)$ Zeit alle cut-vertices in G bestimmt. Begründe die Korrektheit deines Algorithmus und dass er das geforderte Laufzeitverhalten hat. (3 Punkte)

Lösung 2

1. Sei $v \in V$ die Wurzel eines DFS-Baumes von G. Wir zeigen:

   $$v \text{ hat mehr als ein Kind im DFS-Baum} \Leftrightarrow v \text{ ist ein cut-vertex von } G$$

   $\Rightarrow$ Seien $u, w \in V$ zwei Kinder von v im DFS-Baum. Durch das Entfernen von v werden also mindestens die Kanten $\{v, u\}$ und $\{v, w\}$ entfernt. Es kann keine Kante $\{u, w\}$ geben, denn sonst hätte die Tiefensuche einen der beiden Knoten als Nachfolger des anderen und nicht als Nachfolger von v entdeckt. Da G ungerichtet ist, enthält der DFS-Baum keine Querkanten, also keine Kanten zwischen verschiedenen Teilbäumen. Insbesondere ist (u, v, w) der einzige Pfad zwischen u und w in G. Durch das Entfernen von v wird dieser Pfad gelöscht, d.h. G zerfällt in mindestens zwei Zusammenhangskomponenten.

   $\Leftarrow$ Sei v ein cut-vertex von G. Da v die Wurzel des DFS-Baumes ist, enthielte dieser DFS-Baum nur v, wenn v keine Kinder hat. Dann zerfiele G nicht in Zusammenhangskomponenten, wenn v gelöscht wird. Angenommen, v hätte nur ein Kind u im DFS-Baum. Dann ändert sich durch das Entfernen von v nur der Grad von u und G zerfällt nicht in Zusammenhangskomponenten. $\lightning$ Widerspruch zu: v ist ein cut-vertex

2. Sei $v \in V$ ein Knoten, der nicht die Wurzel des DFS-Baumes ist. Wir zeigen:
   Es gibt ein Kind u von v so, dass es keine Rückkante aus dem Teilbaum unter u zu einem Vorgänger von v gibt, genau dann, wenn v ein cut-vertex ist.

   $\Rightarrow$ Wenn es keine Rückkante aus dem Teilbaum unter u zu einem Vorgänger von v gibt, so ist der Pfad von einem beliebigen Vorgänger w von v zu u und jedem beliebigen Nachfolger von u eindeutig. Also ist u nicht mehr von w aus erreichbar, wenn v gelöscht wird. Damit ist v ein cut-vertex.

   $\Leftarrow$ Angenommen, es gibt im Teilbaum unter jedem Kind-Knoten von v im DFS-Baum einen Knoten, der zu einer Rückkante zu einem Vorgänger von v inzident ist. Sei $u \in V$ ein beliebiger Kind-Knoten von v. Dann existiert ein Pfad von u zu einem Vorgänger w von v, der nicht v enthält. Entfernen wir v,

bleibt dieser Pfad also erhalten. Damit bleiben alle Knoten im Teilbaum unter u weiterhin von den Vorgängern von v im DFS-Baum erreichbar. Somit gibt es keine zwei Knoten, die durch das Entfernen von v nicht mehr voneinander erreichbar sind. Damit ist v kein cut-vertex. $\lightning$ Widerspruch

3. Knoten v ist genau dann cut-vertex, wenn es ein Kind u von v gibt, sodass $\text{low}(u) \geq \text{dfs}(v)$.

   $\Rightarrow$ Sei u ein Kind von v mit $\text{low}(u) \geq \text{dfs(v)}$ und sei T_u der Teilbaum von u. Dann existiert keine Rückkante die von einem Knoten aus T_u zu einem Vorfahren von v führt. Demzufolge führen alle Pfade von Vorfahren von v zu einem Knoten in T_u über den Knoten v. Wird v gelöscht gibt es keine Pfade von Vorfahren von v zu Knoten in T_u. Somit ist v ein cut-vertex.

   $\Leftarrow$ Angenommen, es existiert kein Kind u von v mit $\text{low}(u) \geq \text{dfs}(v)$. Also für alle Kinder u von v gilt: $\text{low}(u) < \text{dfs}(v)$. Dann gibt es in jedem zugehörigen Teilbaum T_u eine Rückkante zu einem Vorfahren von v. Demzufolge existiert für jeden Knoten in T_u einen Pfad zu Vorfahren von v, der nicht v enthält. Somit ist v kein cut-vertex.

4. Wir führen zunächst eine Tiefensuche auf G aus und merken uns für jeden Knoten seine DFS-Nummer, seinen low-Wert und seinen Vorgänger im DFS-Baum. Wir zählen nun für jeden Knoten $v \in V$ die Menge und die Anzahl $\text{children}(v)$ seiner Kind-Knoten im DFS-Baum, indem wir die Knoten bestimmen, für die v als Vorgänger gespeichert wurde. Dann überprüfen wir, ob der Knoten s an der Wurzel des DFS-Baumes steht, ein cut-vertex ist, indem wir überprüfen, ob $\text{children}(s) > 1$. Für jeden weiteren Knoten v überprüfen wir, ob er einen Kind-Knoten u hat, sodass $\text{low}(u) \geq \text{dfs(v)}$.
   Wir haben in Teilaufgabe 1 und 2 bewiesen, dass die Bedingung, die unser Algorithmus für einen Knoten $v \in V$ überprüft, äquivalent dazu ist, dass v ein cut-vertex ist. Damit findet unser Algorithmus alle cut-vertices in G.
   Die Tiefensuche iteriert ein Mal über jeden Knoten und jede Kante, d.h. die dafür benötigte Zeit liegt in $\Theta(n + m)$. Wir können dabei sowohl die DFS-Nummern als auch die Vorgänger und low-Werte aller Knoten on-the-fly berechnen. Das Bestimmen der Kind-Knoten für jeden Knoten benötigt Zeit in $\Theta(n)$. Anschießend wird für die Wurzel des DFS-Baumes in konstanter Zeit und für jeden anderen Knoten v in Zeit $\Theta(\text{children}(v))$ eine Bedingung überprüft. Da jeder Knoten nur Kind maximal eines Knoten im DFS-Baum sein kann, benötigt das Überprüfen insgesamt Zeit in $\Theta(n)$. Damit liegt der Gesamtzeitbedarf in $\Theta(n + m)$.

Aufgabe 3 - Handtuch nicht vergessen! (6 Punkte)

Die n Galaxien des Universums sind über m galaktische Hyperraum-Expressrouten verbunden, auf denen sich Reisende zwischen den Galaxien in beide Richtungen bewegen können. Die Reiseagentur von Ursa Minor Beta hat ein besonders gewieftes Preissystem entwickelt: jede*r Reisende kann kostenlos Fahrten unternehmen, die ihn oder sie weiter

von seiner Heimatgalaxie weg bringen. Doch eine Reise, deren Endpunkt näher an der eigenen Heimatgalaxie liegt als ihr Startpunkt, ist exorbitant teuer.
Nachdem ihr Heimatplanet einer unglücklichen Infrastruktur-Maßnahme zum Opfer gefallen ist, beschließt Tricia McMillan, das Universum zu erkunden. Sie ist fest entschlossen, so viele Galaxien wie möglich zu sehen. Ihre Reise beginnt in ihrer Heimatgalaxie, zu der sie letztendlich auch wieder zurück möchte. Leider kann sie sich nur eine einzige Fahrt zu ihrer Heimatgalaxie hin leisten. Deswegen ist ihr Plan, ausschließlich Fahrten von ihrer Heimatgalaxie weg zu unternehmen, bis auf die letzte, die sie wieder zurück zu ihrem Startpunkt bringen soll. Tricia kennt den Abstand jeder Galaxie zu ihrer Heimatgalaxie und weiß außerdem, dass keine zwei Galaxien den gleichen Abstand zu ihrer Heimatgalaxie haben.

1. Beschreibe, wie das Verkehrsnetz zwischen den Galaxien als Graph modelliert werden kann. (1 Punkt)

2. Übersetze Tricias Problem in ein Problem auf deinem Graphen aus Teilaufgabe 1. (1 Punkt)

3. Beschreibe einen Algorithmus, der in $O(n + m)$ Zeit die maximale Anzahl an Galaxien berechnet, die Tricia besuchen kann. Du darfst hierbei annehmen, dass die Galaxien nach Abstand zur Heimatgalaxie sortiert vorliegen. Begründe die Korrektheit deines Algorithmus und warum er die geforderte Laufzeitbedingung einhält. (4 Punkte)
   Hinweis: Die Sortierung ist hilfreich.

Lösung 3

1. Wir modellieren das Verkehrsnetz als ungerichteten, ungewichteten Graphen $G = (V, E)$. Jede Galaxie wird durch genau einen Knoten modelliert, jede Hyperraum-Expressroute zwischen zwei Galaxien als Kante zwischen den entsprechenden Knoten. Zu jedem Knoten merken wir uns, wie weit die zugehörige Galaxie von Tricias Heitmatgalaxie entfernt ist. Für einen Knoten $v \in V$ nennen wir diesen Wert $\text{space}(v)$.

2. Sei $s \in V$ der Knoten in G, der zu Tricias Heitmatgalaxie korrespondiert. Gesucht ist ein Kreis $(s, v_l, v_2, \ldots, v_{k-1}, v_k, s)$ in G, wobei für alle $i \in \{1, \ldots, k-1\}$ gilt, dass $\text{space}(v_i) < \text{space}(v_{i+1})$. Dabei soll k möglichst groß sein.

3. Wir berechnen für jeden Knoten $v \in V$ wird die maximale Länge $\text{travel}(v)$ eines Pfades $(s, v_1, \ldots, v_k = v)$ mit $\text{space}(v_i) < \text{space}(v_{i+1})$ für alle $i \in \{1, \ldots, k-1\}$. Dazu gehen wir wie folgt vor:
   Wir nehmen an, dass die Knoten in aufsteigender Reihenfolge nach ihrem space-Wert vorliegen. Beginnend bei s iterieren wir in aufsteigender Reihenfolge über alle Knoten. Wurde ein Knoten t zu dem Zeitpunkt, an dem er betrachtet wird, nicht in der Nachbarschaft eines zuvor abgearbeiteten Knoten entdeckt, so gibt es

keinen Pfad von s nach t und t wird übersprungen. Ansonsten betrachten wir die Knoten in $u \in \mathrm{N}(t)$ mit $\mathrm{space}(u) > \mathrm{space}(t)$. Gilt $\mathrm{travel}(u) < \mathrm{travel}(t) + 1$ oder $\mathrm{travel}(u)$ wurde noch nicht gesetzt, dann setzen wir $\mathrm{travel}(u) = \mathrm{travel}(t) + 1$ und merken uns t als Vorgänger von u. Haben wir über alle Knoten iteriert, wählen wir einen Knoten $w \in \mathrm{N}(s)$ mit möglichst großem travel-Wert. Der gesuchte Pfad ergibt sich durch Invertieren von $(s, w, \mathrm{parent}(w), \ldots, s)$.
Dieser Algorithmus betrachtet die Knoten in aufsteigender Reihenfolge nach ihren space-Werten, welche paarweise verschieden sind. Damit ist sichergestellt, dass, wenn der Knoten $v \in V$ betrachtet wird, alle Knoten, die in der gesuchten Route von v enthalten sein können, bereits betrachtet wurden. Also ist $\mathrm{travel}(v) = \mathrm{travel}(u) + 1$, wobei u der Vorgänger von v mit dem höchsten travel-Wert ist. Der Algorithmus setzt diesen Wert beim Abarbeiten von u. Damit wählt der Algorithmus zur Konstruktion der Rundreise den Nachbarn von s, der über einen Pfad der größtmöglichen Länge von s aus erreichbar ist.
Es wird zunächst über alle Knoten iteriert. Dabei kann die Nachbarschaft jedes Knoten untersucht werden, d.h. insbesondere wird jede Kante bis zu zwei Mal betrachtet. Dies benötigt Zeit in $\Theta(n + m)$, denn das Aktualisieren der space-Werte benötigt jeweils konstante Zeit. Dann wird in $\Theta(n)$ über alle Nachbarn von s iteriert. Die Konstruktion des Pfades, d.h. das Auslesen der Vorgänger-Knoten und das Invertieren der Knotenabfolge benötigt erneut Zeit in $\Theta(n)$. Insgesamt ergibt sich ein Zeitaufwand in $\Theta(n + m)$.