Karlsruher Institut für Technologie

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

Übungsblatt 12

Algorithmen I – Sommersemester 2022

Abgabe im ILIAS bis 20.07.2022, 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 - Spannend, Bäume! (10 Punkte)

Sei $G = (V, E)$ ein zusammenhängender, gewichteter Graph, bei dem keine zwei Kanten das gleiche Gewicht haben. Wir wollen uns in dieser Aufgabe mit Spannbäumen von G befassen.

1. Sei $C \subset V$ ein Kreis in G. Beweise, dass es keinen minimalen Spannbaum von G gibt, der die schwerste Kante von C enthält. (3 Punkte)

2. Beschreibe einen Algorithmus, der für G einen Spannbaum mit *maximalem* Gewicht bestimmt. Begründe die Korrektheit des Algorithmus und zeige, dass er die asymptotische Laufzeit von Prim / Kruskal nicht überschreitet. (4 Punkte)

3. Wir nennen eine Kantenmenge $X \subset E$ ein *Feedback Edge Set*, wenn G nach Entfernen aller Kanten $x \in X$ azyklisch ist. Gib einen effizienten Algorithmus an, der in G ein Feedback Edge Set mit minimalem Gewicht findet. Begründe außerdem die Korrektheit des Algorithmus und analysiere seine Laufzeit.
Hinweis: die beiden vorherigen Teilaufgaben sind hierfür hilfreich. (3 Punkte)

Lösung 1

1. Sei $T = (V, E_T)$ ein minimaler Spannbaum von G. Wir bezeichnen die schwerste Kante in C als $e_{\max} = \{u, v\}$. Angenommen, es wäre $e_{\max} \in E_T$. Wenn wir $e_{\max}$ aus T entfernen, so zerfällt T in zwei Zusammenhangskomponenten A und B, sodass u und v nicht in der gleichen Zusammenhangskomponente liegen. Betrachte nun den Pfad π von u nach v der entsteht, wenn $\{u, v\}$ aus C entfernt wird. Da u und v in unterschiedlichen Komponenten liegen, gibt es mindestens eine Kante $e = \{u', v'\}$ auf π, sodass $u' \in A$ und $v' \in B$. Durch Hinzufügen von e erhalten wir $T' = (V, E'_T)$ mit $E'_T = (E_T \backslash \{e_{\max}\}) \cup \{e\}$. Dabei ist T' ein Baum, denn $|E_T| = n - 1$ und somit $|E'_T| = n - 1$ und durch Hinzunahme von e werden die beiden Komponenten A und B zu einer Zusammenhangskomponente verbunden, d.h. T' ist zusammenhängend und hat $n - 1$ Kanten. Da $e_{\max}$ die schwerste Kante auf C ist und alle Kantengewichte verschieden sind, ist e leichter als $e_{\max}$. Damit ist T' ein Spannbaum von G, dessen Gesamtgewicht der Kanten kleiner als das der Kanten in T ist. $\lightning$ Widerspruch zu: T ist minimaler Spannbaum von G

2. Wir können den Algorithmus von Prim verwenden, den wir bereits in der Vorlesung kennengelernt haben. Dazu passen wir die Kantengewichtung von G wie folgt an: Zunächst bestimmen wir das Gewicht $w_{\max}$ der schwersten Kante in G. Anschließend weisen wir jeder Kante e mit Gewicht $w(e)$ das neue Gewicht $w'(e) = w_{\max} - w(e)$ zu. Auf G mit der Kantengewichtung w' rufen wir nun den Algorithmus von Prim auf. Die Kanten, die uns dieser zurückgibt, bilden einen maximalen Spannbaum von G.
   Sei $T = (V, E_T)$ der Spannbaum von G, der auf diese Weise bestimmt wurde. Wir zeigen, dass T tatsächlich maximales Gewicht hat:
   Angenommen, es gäbe einen Spannbaum $T' = (V, E_{T'})$ von G mit größerem Gewicht, es ist also $\sum_{e \in E_T} w(e) < \sum_{e \in E_{T'}} w(e)$. Dann:

$$\begin{aligned} w'(T) &= \sum_{e \in E_T} w'(e) \\ &= \sum_{e \in E_T} \Big(w_{\max} - w(e) \Big) \\ &= (n-1) w_{\max} - \sum_{e \in E_T} w(e) \\ &> (n-1) w_{\max} - \sum_{e \in E_{T'}} w(e) \\ &= w'(T') \end{aligned}$$

   $\lightning$ Widerspruch zu: T ist MST in G mit Kantengewichtung w'.
   Zur Bestimmung von $w_{\max}$ müssen wir in $\Theta(m)$ über alle Kanten iterieren. Das Anpassen der Kantengewichtung benötigt ebenfalls Zeit in $\Theta(m)$. Anschließend führen wir den Algorithmus von Prim aus, dessen asymptotische Laufzeit in $\Theta(n \log(n) +$

m) liegt. Damit ergibt sich eine Gesamtlaufzeit in $\Theta(n \log(n) + m)$.

3. Wir verwenden den Algorithmus aus Teilaufgabe 2, um einen maximalen Spannbaum T von G zu bestimmen. Die Kanten in G, die nicht in diesem Spannbaum enthalten sind, bilden ein Feedback Edge Set von G. Wir nennen die Menge dieser Kanten X.
   Dieser Algorithmus bestimmt ein Feedback Edge Set, denn: Jede Kante in X ist leichteste Kante auf einem Kreis. Dies lässt sich analog zu Teilaufgabe 1 zeigen. Da T ein Spannbaum ist, muss G nach Entfernen der Kanten aus X azyklisch sein. Zudem schließt jede dieser Kanten einen Kreis mit Kanten von T, d.h. jeder Graph, der aus G durch Entfernen einer echten Teilmenge von Kanten aus X entsteht, ist nicht azyklisch. X hat minimales Gewicht, da aus jedem Kreis in G mindestens eine Kante entfernt werden muss, damit der resultierende Graph azyklisch ist, und wir dazu die leichtesten Kanten gewählt haben.
   Die Laufzeit dieses Algorithmus entspricht der Laufzeit unseres Algorithmus aus Teilaufgabe 2 und damit der des Algorithmus von Kruskal.

Aufgabe 2 - Darf's ein bisschen weniger sein? (3 Punkte)

Gegeben sei ein zusammenhängender, gewichteter Graph $G = (V, E)$ und ein Knoten $s \in V$. Gib einen Algorithmus an, der für jeden Knoten $t \in V$ das minimale Kantengewicht w_t bestimmt, sodass es einen s-t-Pfad gibt, bei der jede Kante ein Gewicht von höchstens w_t hat. Begründe die Korrektheit deines Algorithmus. Nenne und begründe außerdem seine asymptotische Laufzeit.

Lösung 2

Wir verwenden zunächst den Algorithmus von Prim, um einen MST T von G zu bestimmen. Anschließend bestimmen wir für jeden Knoten $t \in V$ das Gewicht $\texttt{upper}(t)$ der schwersten Kante auf dem (eindeutigen) s-t-Pfad in T.
Für jeden Knoten $v \in V$ ist der von diesem Algorithmus berechnete Wert $\texttt{upper}(v)$ genau w_v. Angenommen, es gäbe einen Knoten $t \in V$, für den $\texttt{upper}(t) \neq w_t$, da w_t minimal also $\texttt{upper}(t) > w_t$. Sei e die Kante mit Gewicht $\texttt{upper}(t)$ auf dem Pfad zwischen s und t in T und sei e' die Kante mit Gewicht w_t wie in der Aufgabenstellung. Dann gibt es also einen Pfad zwischen s und t, der nicht die e, aber dafür e' enthält. Also gibt es einen Kreis in G, auf dem sowohl e als auch e' liegen. Durch Entfernen von e und Hinzufügen von e' zu T erhalten wir einen Spannbaum T' von G, dessen Gewicht kleiner ist als das von T. $\lightning$ Widerspruch zu: T ist MST von G.
Die asymptotische Laufzeit des Algorithmus von Prim liegt in $\Theta(n \log(n) + m)$. Zum Bestimmen der `upper`-Werte kann eine Breiten- oder Tiefensuche in Zeit $\Theta(n + m)$ genutzt werden. Damit liegt die Laufzeit unseres Algorithmus in $\Theta(n \log(n) + m)$.

Aufgabe 3 - Amortisierte Analyse (7 Punkte)

Wir betrachten eine Datenstruktur, die zur Verwaltung von Elementen genutzt wird. Dabei wird zwischen roten und blauen Elementen unterschieden. Wir bezeichnen mit n_b die Anzahl blauer und mit n_r die Anzahl roter Elemente in der Datenstruktur. Es stehen die folgenden Operationen zur Verfügung:

- ADD(): Einfügen eines blauen Elements in $\Theta(1)$
- MARK(): Umfärben der aufgerundeten Hälfte der blauen Elemente auf rot in $\Theta(n_b)$
- REMOVE(): Löschen aller roten Elemente in $\Theta(n_r)$

1. Gib für die folgenden beiden Abfolgen von Operationen an die Gesamtkosten an:

$$k \cdot \text{ADD}, \text{MARK}, \text{REMOVE}$$

$$m \cdot (k \cdot \text{ADD}, \text{MARK})$$

(2 Punkte)

2. Zeige mit Hilfe der Kontomethode, dass in jeder beliebigen Abfolge von Operationen jede Operation amortisiert konstante Kosten hat. (2 Punkte)

3. Zeige mit Hilfe der Potentialmethode, dass in jeder beliebigen Abfolge von Operationen jede Operation amortisiert konstante Kosten hat. (3 Punkte)

Lösung 3

1. Die Operation ADD hat konstante Kosten, MARK konstante Kosten pro blauem Element und REMOVE konstante Kosten pro rotem Element. Damit:
   - Die k-malige Anwendung von ADD hat Kosten k. Anschließend ist $n_b = k$, das bedeutet MARK kostet ebenfalls k und setzt $n_r = \lceil k/2 \rceil$, weswegen auch REMOVE Kosten $\lceil k/2 \rceil$ hat. Die Gesamtkosten liegen damit bei $\lceil 5k/2 \rceil$.
   - Sei $0 \leq i \leq m$. Nach der i-ten Durchführung von k-Mal ADD und ein Mal MARK ist $n_b = \frac{(2^i-1)k}{2^i}$. Nach m Durchläufen befinden sich also $m \cdot k$ Elemente in der Datenstruktur, von denen $\frac{(2^m-1)k}{2^m}$ blau sind. Also beträgt die Anzahl roter Elemente

$$mk - \frac{(2^m-1)k}{2^m} = \frac{2^m mk - (2^m-1)k}{2^m} = \frac{2^m mk - 2^m k + k}{2^m} = (m-1)k + \frac{k}{2^m}$$

   Jedes rote Element in der Datenstruktur wurde genau ein Mal via ADD eingefügt und via MARK umgefärbt. Damit sind pro rotem Element in der Datenstruktur Kosten 2 entstanden und die gesamte Abfolge benötigt Kosten

$$mk + 2(m-1)k + \frac{2k}{2^m} = mk + (2m-2)k + \frac{k}{2^{m-1}} = (3m-2)k + \frac{k}{2^{m-1}}$$

2. Die tatsächlichen Kosten von ADD liegen bei einem Token, die von MARK bei n_b und die von REMOVE bei n_r. Wir zahlen pro ADD-Operation nun zusätzlich drei Tokens ein. Die amortisierten Kosten von ADD liegen also bei $4 \in \Theta(1)$. Sowohl MARK als auch REMOVE heben pro blauem bzw. rotem Element ein Token ab.
   Die Anzahl an blauen Elementen wird bei einem Aufruf von MARK mindestens halbiert. Damit braucht es stets höchstens $\log(n_b)$ Aufrufe von MARK, um sämtliche blauen Elemente umzufärben. Also entfallen auf jedes blaue Element maximal zwei Token. Vor jedem Aufruf von REMOVE befinden sich also mindestens n_r Tokens auf dem Konto. Damit liegen die amortisierten Kosten von sowohl MARK als auch REMOVE bei $0 \in \Theta(1)$. Damit wird der Kontostand nie negativ und jede Operation hat amortisiert konstante Kosten.

3. Sei D unsere Datenstruktur. Wir bezeichnen mit D_{vor} und D_{nach} den Zustand von D vor bzw. nach einer Operation. Die tatsächlichen Kosten von ADD liegen bei 1, die von MARK bei n_b und die von REMOVE bei n_r. Als Potentialfunktion wählen wir uns
$$\phi(D) = 3n_b + n_r$$
   Nun zeigen wir, dass für jede der drei Operationen die amortisierten Kosten in $\Theta(1)$ liegen:

   - ADD: n_b wird um 1 erhöht. Damit ist $\phi(D_{\text{nach}}) - \phi(D_{\text{vor}}) + 3$ und die amortisierten Kosten von ADD liegen bei
   $$1 + 3 = 4 \in \Theta(1)$$

   - MARK: Durch MARK werden $\lceil n_b/2 \rceil$ Elemente umgefärbt, d.h. n_b wird um $\lceil n_b/2 \rceil$ verringert, n_r um den gleichen Wert erhöht. Also ist
   $$\begin{aligned}\phi(D_{\text{nach}}) - \phi(D_{\text{vor}}) &= 3(n_b - \lceil n_b/2 \rceil) + n_r + \lceil n_b/2 \rceil - 3n_b - n_r \\ &= \lceil n_b/2 \rceil - 3\lceil n_b/2 \rceil \\ &= -2\lceil n_b/2 \rceil\end{aligned}$$
   und wir erhalten amortisierte Kosten von
   $$n_b - 2\lceil n_b/2 \rceil \in \Theta(1)$$

   - REMOVE: Da alle roten Elemente gelöscht werden, liegen die amotisierten Kosten von REMOVE bei
   $$n_r + \phi(D_{\text{nach}}) - \phi(D_{\text{vor}}) = n_b + 0 - n_b - n_r = n_r - n_r = 0 \in \Theta(1)$$