Karlsruher Institut für Technologie

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

Übungsblatt 06

Algorithmen I - Sommersemester 2024

Abgabe im ILIAS bis 07.06.2024, 18:00 Uhr

Die Abgabe erfolgt alleine oder zu zweit als *eine* PDF-Datei über das Übungsmodul in der Gruppe eures Tutoriums im ILIAS. Beschriftet die Abgabe deutlich mit Matrikelnummer und Name.

- Achtet bei handschriftlichen Abgaben auf Lesbarkeit.
- Achtet darauf, ob Algorithmen in Worten oder Pseudocode beschrieben werden sollen.
- Es werden immer asymptotisch möglichst effiziente Algorithmen erwartet, wenn nicht anders angegeben.
- Wenn Korrektheits- oder Laufzeitanalysen gefordert sind, behandelt diese separat von der Algorithmenbeschreibung.

Gesamtpunkte: 20 (+ 2 Bonus)

Aufgabe 1 - Weil Baum! (7 Punkte)

Da Bäume wichtige und grundlegende Graphen sind, sollst du in dieser Aufgabe Bäume und ein paar ihrer Eigenschaften näher kennen lernen. Wir beschränken uns dabei auf Graphen mit mindestens einem Knoten.

a) Zeige, dass jeder Baum mindestens ein Blatt, also einen Knoten mit Grad höchstens 1, enthält. (1.5 Punkte)

b) Sei T ein Graph mit n Knoten. Gegeben sind die folgenden drei Definitionen eines Baumes:

   1) T ist nach der Definition aus der VL ein Baum, also: T ist zusammenhängend und kreisfrei,
   2) T ist kreisfrei und hat $n - 1$ Kanten,
   3) T ist zusammenhängend und hat $n - 1$ Kanten.

   Zeige, dass die drei Definitionen äquivalent sind. (5.5 Punkte)

Lösung 1

a) Angenommen, es gäbe einen Baum $T = (V, E)$ mit $|V| = n$, der kein Blatt enthält. Dann können wir uns einen beliebigen Knoten $v \in V$ wählen und von v aus T traversieren. Da nach Annahme jeder Knoten mindestens Grad 2 hat, finden wir auch für jeden Knoten $u \in V$ einen Nachfolger $w \in N(u)$, der nicht der Knoten ist, von dem aus u gefunden wurde. Nach maximal n Schritten stoßen wir auf einen bereits besuchten Knoten, da T nur n Knoten enthält. Somit hat T einen Kreis. $\lightning$ Widerspruch zu: T ist kreisfrei
Also enthält T ein Blatt.

b)

- Wir beweisen zunächst 1) $\Rightarrow$ 2), also dass ein zusammenhängender, kreisfreier Graph auch $n-1$ Kanten hat (dass dieser Graph kreisfrei ist, ist per Definition gegeben):
  Sei $G = (V, E)$ ein Baum, d.h. G ist zusammenhängend und kreisfrei. Wir zeigen mittels vollständiger Induktion über n, dass G genau $n - 1$ Kanten hat.

  IA:
  - $n = 1$ Der einzige kreisfreie Graph mit nur einem Knoten hat keine Kanten, also muss $m = n - 1 = 0$ sein. ✓
  - $n = 2$ Der einzige zusammenhängende, kreisfreie Graph mit zwei Knoten und ohne Doppelkanten enthält genau eine Kante, die die beiden Knoten verbindet, also muss $m = n - 1 = 1$ sein. ✓

  IV: Sei $n \in \mathbb{N}$ fest, aber beliebig. Ein Baum mit n Knoten enthält $n - 1$ Kanten.

  IS: $n \rightsquigarrow n + 1$
  Sei T ein Baum mit $n+1$ Knoten. Dann enthält T ein Blatt v. Entfernen wir v und die zu v inzidente Kante aus T, enthalten wir einen Graphen T' mit n Knoten. Dieser muss ebenfalls zusammenhängend sein, da v ein Blatt in T ist. Zudem muss T' kreisfrei sein, weil T kreisfrei ist. Also hat T' nach **IV** genau $n - 1$ Kanten. Damit hat T genau $n = n + 1 - 1$ Kanten.

- Als nächstes zeigen wir 2) $\Rightarrow$ 3), also dass ein kreisfreier Graph mit $n - 1$ Kanten auch zusammenhängend ist. Die zweite Eigenschaft, $n - 1$ Kanten zu haben, folgt per Definition.
  Sei $G = (V, E)$ kreisfrei und sei $m = |E| = n - 1$. Angenommen, G ist nicht zusammenhängend, also ein Wald. Dann setzt sich G aus k Bäumen zusammen, wobei $2 \leq k \leq n$. Wie wir oben gezeigt haben, folgt daraus, dass jeder dieser Bäume eine Kante weniger als Knoten enthält. Damit kann G höchstens $n - 2$ Kanten enthalten. $\lightning$ Widerspruch zu: G enthält $n - 1$ Kanten.

- Zuletzt zeigen wir (3) $\Rightarrow$ 1)), also: T ist zusammenhängend und hat $n - 1$ Kanten $\Rightarrow$ T ist kreisfrei. Hier können wir den Zusammenhang übernehmen.

Sei $G = (V, E)$ zusammenhängend und sei $m = |E| = n - 1$. Angenommen, G ist nicht kreisfrei. Dann enthält G einen Kreis C mit k Knoten, wobei $3 \leq k \leq n$. Auf diesem Kreis liegen k Kanten. Andererseits liegen $n - k$ Knoten in G nicht auf C. Da G zusammenhängend ist, gibt es von jedem solchen Knoten einen kürzesten Pfad zu einem der Kreisknoten. Betrachtet man die jeweils erste Kante all dieser Pfade, so erhält man $n-k$ verschiedene Kanten, die nicht auf dem Kreis liegen. Zusammen mit den k Kanten auf dem Kreis erhalten wir $m \geq n$ Kanten. $\lightning$ Widerspruch zu: $m = n - 1$.

Aufgabe 2 - Die Musterlösung hat heute eine Verspätung von ca. 10 Tagen (8 Punkte)

Um seinen Geheimplan umsetzen zu können, muss Dr. Meta im nächsten Schritt seine Biber an strategisch wichtigen Punkten in ganz Deutschland verteilen. Dafür möchte er mithilfe des Deutschlandtickets seine Biber mit dem Zug reisen lassen. Beim Besuch der Vorlesung Algorithmen I hat Dr. Meta gehört, dass man mithilfe von Breitensuche kürzeste Wege berechnen kann. Allerdings kam ihm just in diesem Moment eine Idee, wie er das [kœri]-Werk stilllegen kann und daher hat er die genauen Details nicht mitbekommen. Aufgrund dessen beauftragt er seine besten Biber mit der Erforschung des Themas.

Diese modellieren das Schienennetz als zusammenhängende und ungerichtete Graphen. In der Vorlesung haben wir gesehen, dass Breitensuche in ungewichteten Graphen kürzeste Pfade von einem Startknoten s zu allen anderen Knoten findet. In den ersten beiden Teilaufgaben wollen wir uns nochmal *ungewichtete* Graphen genauer anschauen.

a) Wir repräsentieren die Knoten eines Graphen G durch die Zahlen 0 bis $n-1$, wobei n die Anzahl der Knoten ist. Nun wollen wir das Array bestimmen, das für jeden Index t die Distanz $\text{dist}(s, t)$ von Startknoten s zu Knoten t enthält. Beschreibe, wie du die Breitensuche anpassen musst, um dieses Array berechnen zu können. Die Laufzeit der Breitensuche soll sich dabei nicht ändern. (2 Punkte)

b) Nun wollen wir nicht nur die Distanz von s zu einem Knoten t bestimmen, sondern zusätzlich auch den kürzesten Pfad. Beschreibe, welche Informationen man sich zusätzlich zum Distanzarray noch während der BFS speichert und wie man damit dann den kürzesten Pfad von s zu t rekonstruiert. Auch hier soll sich die Laufzeit der Breitensuche nicht ändern. (2 Punkte)

Bei der Modellierung des Schienennetzes haben die Biber allerdings die Fahrtdauer in Stunden auf einer Strecke als ganzzahliges und positives Kantengewicht (> 0) verwendet. Wir betrachten also nun *gewichtete* Graphen.

c) Gib einen Beispielgraphen mit Kantengewichten und Knoten s und t an, bei dem die Breitensuche aus der Vorlesung nicht den kürzesten Pfad zwischen s und t findet. Begründe. (0.5 Punkte)

Auf der Suche nach einem Ausweg aus der Klemme, steckt dir die Agentin Johnson, die Dr. Meta in den Bahnvorstand eingeschleust hat, folgende brisante Information zu: Im Schienennetz der Deutschen Bahn braucht ein Zug von einem Bahnhof zum nächsten höchstens k Stunden.

d) Beschreibe nun einen Algorithmus, der mithilfe von Breitensuche in einem Graphen, in dem jede Kante höchstens Gewicht k hat, die kürzesten Pfade vom Startknoten s zu allen anderen Knoten bestimmt. (2 Punkte)
Tipp: Modifiziere den Graphen so, dass die Breitensuche das korrekte Ergebnis liefert.

e) Da die Zeit drängt, musst du Dr. Meta zeigen, dass dein Algorithmus ausreichend schnell ist. Gib hierfür die Laufzeit deines Algorithmus in Abhängigkeit von k, Knotenzahl n und Kantenzahl m an. Begründe diese. (1 Punkt)

f) Wann stößt dein Algorithmus auf Probleme und wird auf auch auf gewichteten Graphen mit wenigen Knoten bzw. Kanten zu langsam? (0.5 Punkte)

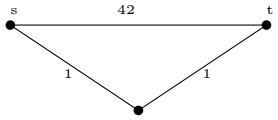
Es meldet sich bei dir ein mysteriöser Agent mit dem Decknamen Dijkstra. *To be continued...*

Lösung 2

a) Um die Distanzen zu berechnen, gehen wir analog zur Breitensuche der Vorlesung vor (siehe Pseudocode in Vorlesung 8, Folie 8). Wir speichern uns initial für unseren Startknoten s die Distanz 0 in unserem Array.
Die Distanz $\text{dist}(s, t)$ zwischen dem Startknoten s und einem Knoten t entspricht der Lage, in der der Knoten t gefunden wurde. Ein Knoten t liegt daher immer eine Lage tiefer als der Knoten, von dem aus t gefunden wurde. Falls wir also einen neuen Knoten u aus der Nachbarschaft des Knoten v in die Queue einfügen, so setzen wir zusätzlich $\text{dist}[u] = \text{dist}[v]+1$ in unserer Datenstrukur. Da wir Knoten v bereits gesehen und somit die Distanz bereits berechnet haben, ist dieses Ergebnis wohldefiniert.
Die Laufzeit der Breitensuche ändert sich dabei nicht, da wir in jedem Schritt nur eine Addition ausführen und konstant oft auf Arrays zugreifen.

b) Zusätzlich speichern wir uns noch für jeden Knoten v den Knoten u, von dem aus v während der BFS zuerst entdeckt wird, in einem Array. Dann gibt es einen kürzesten Weg von s nach v, bei dem u der Vorgänger von v auf diesem Pfad ist. Wenn wir, bei t beginnend, iterativ immer zum Vorgänger des aktuellen Knotens laufen, bis wir bei s angekommen sind, traversieren wir den kürzesten Pfad in umgekehrter Reihenfolge. Wir erhalten also den kürzesten Pfad von s nach t, indem wir die Liste mit den dabei besuchten Knoten umdrehen.

c) Der untenstehende Graph ist eine mögliche Lösung.



Die Breitensuche findet den Knoten t von s aus, aber der kürzeste Weg von s zu t hat Länge 2.

d) Zuerst gehen wir über alle Kanten im Graph und ersetzen eine Kante mit Gewicht w durch w Kanten mit Gewicht 1. Im Detail bedeutet dies für die Kante $\{u, v\}$: Wir entfernen die Kante $\{u, v\}$ und fügen dann $w-1$ neue Knoten ein. Dann verbinden wir u mit dem ersten neuen Knoten und v mit dem letzten. Abschließend verbinden wir die neuen Knoten zu einem einfachen Pfad.
Nun haben wir einen ungewichteten Graphen und können die Breitensuche aus der Vorlesung ausführen.

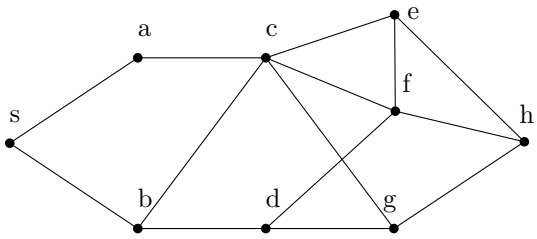
e) Um eine Kante mit Gewicht w zu bearbeiten, müssen wir eine Kante entfernen, $w-1$ neue Knoten und w neue Kanten einfügen. Da alle Kantengewichte kleiner gleich k sind, dauert dies maximal $\mathcal{O}(k)$ Zeit pro Kante und somit insgesamt maximal $\mathcal{O}(m \cdot k)$. Der entstehende Graph hat dann maximal $n + m \cdot (k-1)$ Knoten und $k \cdot m$ Kanten. Somit ist die Laufzeit der Breitensuche in $\mathcal{O}(n + m \cdot k)$.

f) Bei großen Kantengewichten. Diese können potentiell deutlich größer als n sein.

Aufgabe 3 - Wie viele Wege führen nach Rom? (5 Punkte)

Gegeben sei ein ungerichteter und ungewichteter Graph. Wir wollen uns in dieser Aufgabe damit beschäftigen, wie viele kürzeste Pfade es jeweils zwischen zwei Knoten gibt. Zwei Pfade sind verschieden, wenn sie sich in mindestens einem Knoten unterscheiden.

a) Gegeben sei der folgende Beispielgraph. Gib für jeden der Knoten a, b, c, d, e, f, g, h an, wie viele kürzeste Wege es zwischen s und diesem Knoten gibt. (1 Punkt)



b) Beschreibe eine Breitensuchvariante, die bestimmt, wie viele kürzeste Pfade es zwischen einem gegebenen Knoten s und jedem anderen Knoten gibt. Gib außerdem die Laufzeit des Algorithmus abhängig von Knotenzahl n und Kantenzahl m an und begründe diese. Begründe auch die Korrektheit deines Ansatzes. (4 Punkte)

Lösung 3

a) Es ergeben sich die folgenden Anzahlen:

- a: 1
- b: 1
- c: 2
- d: 1
- e: 2
- f: 3
- g: 3
- h: 8

Zu x und u gibt es jeweils eine Kante von s und somit einen Pfad der Länge eins. Zu y gibt es die Pfade (s, x, y) und (s, u, y) der Länge zwei. Zu v gibt es die Pfade (s, x, v) und (s, u, v) der Länge zwei. Zu t und w können wir je über v oder y gehen und erhalten so vier Pfade der Länge drei. Zu r können wir nur über y gehen und erhalten daher zwei Pfade der Länge drei. Zu z können wir je über t oder w gehen und erhalten so acht Pfade der Länge vier.

b) Die Idee ist eine Breitensuche auszuführen und dabei die Anzahl der kürzesten Pfade mitzuberechnen. Wir speichern dafür ein Array *paths* mit der aktuell gefundenen Anzahl an kürzesten Pfaden. Dieses initialisieren wir mit 1 für s und sonst mit 0. Wenn wir nun von Knoten u in Layer l aus den Knoten v betrachten, dann unterscheiden wir je nach Layer von v zwei Fälle:

1. v ist in einem Layer kleiner gleich l. Dann haben wir einen längeren Pfad gefunden und müssen nichts tun.
2. v ist in Layer $l + 1$ (oder wird dort neu eingefügt), dann haben wir neue kürzeste Pfade gefunden und müssen diese zu den bestehenden hinzufügen. Wir berechnen also $paths[v] = paths[v] + paths[u]$

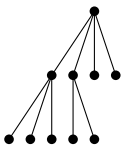
Da wir in jedem Schritt nur eine zusätzliche Addition ausführen, bleibt die Laufzeit im Vergleich zur klassischen Breitensuche gleich. Also ist diese $\Theta(n + m)$.

Wir zeigen die Korrektheit wie folgt: Kanten, die nicht in die nächste Schicht führen, geben uns keine neuen kürzesten Pfade. Die entstehenden Pfade sind nämlich nicht optimal, da wir zu Knoten aus vorherigen Schichten bereits kürzere Pfade gefunden haben. Im Stile einer Induktion folgt, dass alle kürzesten Pfade zu einem Knoten u, der eine Kante zu einem Knoten v hat, auch zu kürzesten Pfaden für v werden, wenn v in der nächsten Schicht von u ist. Somit ist eine Addition im Fall von kürzesten Pfaden korrekt.
Da wir jede Kante nur einmal betrachten, werden auch keine Pfade doppelt gezählt.

Aufgabe 4 - Zum Knobeln (0 Punkte + 2 Bonus)

Dr. Meta befürchtet, dass seine Biber vor lauter Bäumen den Wald nicht mehr sehen. Daher sollst du deinen guten Willen unter Beweis stellen und den Bibern einige Bäume zeigen. Dr. Meta wünscht sich, dass die von dir gezeichneten Bäume klein sind. Daher hat er sich auf eine Knotenzahl von **genau** 10 festgelegt. Außerdem mag er die Zahl zwei nicht, daher möchte er in deinen Beispielen keine Knoten mit Grad zwei finden. Dr. Metas finaler Wunsch ist, dass die Bäume auch wirklich verschieden, also nicht isomorph[1] sind.

Zeichne alle Bäume, die Dr. Metas Spezifikation erfüllen. Gib zusätzlich für jeden Baum seine Gradsequenz in absteigender Reihenfolge an.



$[4, 4, 3, 1, 1, 1, 1, 1, 1, 1]$

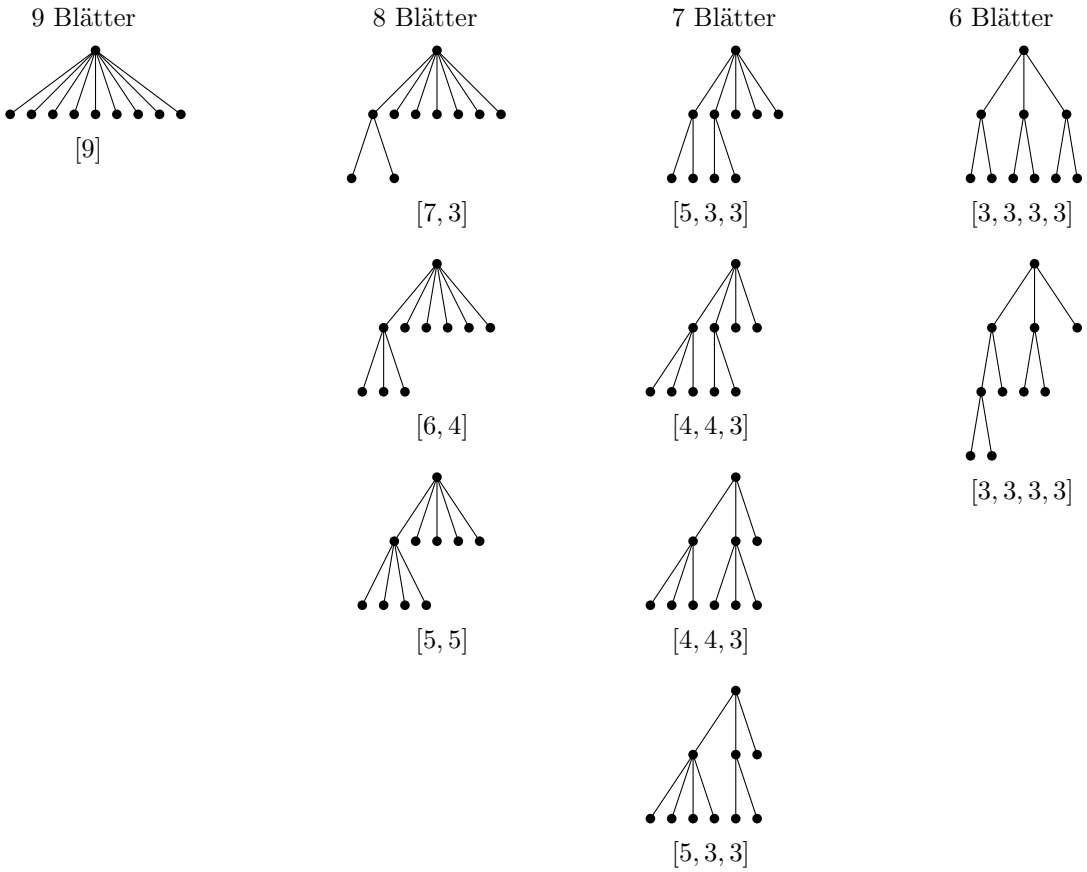
Beispiel: für einen Baum, der die Spezifikation erfüllt mit seiner Gradsequenz. Der Baum enthält zwei Knoten mit Grad 4, einen Knoten mit Grad 3 und sieben Knoten mit Grad 1.

(2 Punkte)

Hinweis: Die Gesamtanzahl der Bäume ist ausreichend klein, dass du diese problemlos zeichnen kannst.

[1] Zwei Graphen $G = (V_G, E_G)$ und $H = (V_H, E_H)$ sind isomorph, falls sie die gleiche Struktur haben. Formal sind G und H isomorph, falls es eine Abbildung $f : V_G \to V_H$ gibt, sodass $\forall u, v \in V$ gilt: $\{u, v\} \in E_G$ gdw. $\{f(u), f(v)\} \in E_H$.

Lösung 4



Alle Graphen, die Dr. Metas Spezifikation entsprechen. Die Zahlenfolge in den eckigen Klammern gibt die Gradsequenz (ohne Blätter) an.

Dieses Problem ist durch den Film **Good Will Hunting** bekannt. Das folgende Video beschäftig sich mit dem Problem `https://www.youtube.com/watch?v=iW_LkYiuTKE`