Karlsruher Institut für Technologie

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

Übungsblatt 06

Algorithmen I – Sommersemester 2022

Abgabe im ILIAS bis 01.06.2022, 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!

Aufgabe 1 - Ich bin Groot! (8 Punkte)

In der Vorlesung wurde definiert, dass ein Baum ein zusammenhängender und kreisfreier Graph ist. Tatsächlich haben Bäume noch weitere spannende Eigenschaften, von denen wir einige in dieser Aufgabe kennenlernen.
Wir beschränken uns dabei auf Graphen mit mindestens einem Knoten.

1. Zeige, dass jeder Baum mindestens ein Blatt, also einen Knoten mit Grad höchstens 1, enthält. (2 Punkte)

2. Sei T ein Graph mit n Knoten. Angenommen, T hat zwei der folgenden Eigenschaften

   a) T ist zusammenhängend,

   b) T ist kreisfrei,

   c) T hat $n-1$ Kanten,

   Zeige, dass T dann auch die dritte Eigenschaft hat. (6 Punkte)

Lösung 1

1. 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.

2. Wir beweisen zunächst, dass aus den Eigenschaften a) und b), also der Charakterisierung von Bäumen aus der Vorlesung, die Eigenschaft c) folgt:
   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.

   Nun zeigen wir noch die übrigen beiden Implikationen:

   a) b) und c) $\Rightarrow$ a)
   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.

   b) a) und c) $\Rightarrow$ b)
   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. Damit liegen $n - k$ Knoten in G nicht auf C. Diese müsste n mit den übrigen $n - 1 - k$ Kanten so zu den Knoten auf C verbunden sein, dass G zusammenhängend ist. Das ist nicht möglich. $\lightning$ Widerspruch zu: G ist zusammenhängend

Aufgabe 2 - **BFS** (12 Punkte)

Die Breitensuche (BFS) ist ein mächtiges Werkzeug zum Explorieren eines Graphen. In der Vorlesung wurde Pseudocode vorgestellt, um mittels einer BFS die Knoten eines Graphen einzufärben (siehe Vorlesung 8, Folie 8, *Pseudocode (Endergebnis)* `BFS(...)`). Im Folgenden sollen Adaptionen der BFS entwickelt werden, die ihre asymptotische Laufzeit nicht verändern.
Hinweis: Du darfst annehmen, dass die n Knoten in einem Graphen durch die Zahlen $0, \ldots, n-1$ repräsentiert werden. Achte darauf in welcher Aufgabe Pseudocode gefordert ist und wo Algorithmen textuell beschrieben werden sollen.

1. Gib aufbauend auf den `BFS(...)` Pseudocode einen Algorithmus im Pseudocode an, der einen ungerichteten und ungewichteten Graphen G zusammen mit einem Knoten s erhält und eine Datenstruktur ausgibt, aus der man für jeden Knoten t in Zeit $O(1)$ die Distanz $\text{dist}(s,t)$ in G ablesen kann. Beschreibe welche Datenstruktur du wählst und welcher Wert in einem gegebenen Eintrag gespeichert wird.
   (3 Punkte)

2. Anstatt der Distanz zwischen zwei Knoten s und t soll nun ein kürzester Pfad zwischen ihnen bestimmt werden. Beschreibe einen Algorithmus, der einen gerichteten und ungewichteten Graph G zusammen mit zwei Knoten s und t erhält und aufbauend auf eine BFS einen kürzesten Pfad von s nach t bestimmt. Beschreibe dafür zuerst, welche Hilfsdatenstruktur du dazu wählst und welcher Wert in einem gegebenen Eintrag gespeichert wird. Beschreibe einen weiteren Algorithmus der im Nachhinein in $O(n)$ den kürzesten Pfad aus deiner Hilfsdatenstruktur rekonstruiert. (4 Punkte)

3. Wir wollen nun untersuchen, ob ein Graph G *bipartit* ist. Das ist genau dann der Fall, wenn man seine Knotenmenge in zwei Teilmengen zerlegen kann, sodass zwischen den Knoten innerhalb einer Menge keine Kanten verlaufen.
   Aufbauend auf den `BFS(...)` Pseudocode, gib einen Algorithmus im Pseudocode an, der einen ungerichteten und ungewichteten Graphen G erhält und ausgibt, ob G bipartit ist. Begründe die Korrektheit deines Algorithmus.
   Hinweis: Können benachbarte Knoten in der gleichen Partition sein? Beachte, dass sich hier die Eingabe des Algorithmus von dem im `BFS(...)` Pseudocode unterscheidet.
   (5 Punkte)

Lösung 2

1. Idee: Wir verwenden ein Array mit einem Eintrag pro Knoten. Im Eintrag zu $t \in V$ speichern wir die Lage, in der t gefunden wurde. Diese entspricht der Anzahl an Kanten, die auf einem kürzesten Pfad von s nach t liegen und damit $\text{dist}(s,t)$. Das Ergebnis steht dann im Arrayeintrag von t. Ein Knoten t liegt immer eine Lage tiefer als der Knoten, von dem aus t gefunden wurde.

```
1:  DISTANCEBFS(G = ({v_0, ..., v_{n-1}}, E) : Graph, s ∈ V) : [ℕ; n]
2:      dist : [ℕ; n] = ⟨0, ..., 0⟩
3:      Q : Queue
4:      Q.PUSH(s)
5:      COLOR(s)
6:      while ¬Q.empty do
7:          v_i : Node = Q.POP()
8:          for v_k ∈ N(v_i) do
9:              if UNCOLORED(v_k) then
10:                 Q.PUSH(v_k)
11:                 dist[k] = dist[i] + 1
12:                 COLOR(v_k)
13:             end
14:         end
15:     end
16:     return dist
```

2. Idee: Wir wissen, dass die BFS in ungewichteten Graphen kürzeste Wege bestimmt. Also bilden die Knoten, über die die BFS einen Knoten t von s aus findet, einen kürzesten Pfad von s nach t.
   Diese Variante der BFS soll ein Array `parent` mit einem Eintrag pro Knoten anlegen. Im Eintrag zum Knoten $v \in V$ soll der Knoten stehen, über den v von der BFS gefunden wurde. Dazu können wir unsere BFS-Variante aus Teilaufgabe 1 anpassen: Anstatt dem Array `dist` wird das Array `parent` verwendet. Dieses wird mit $\perp$-Einträgen initialisiert. Bevor ein Knoten v_i, der in der Nachbarschaft von u gefunden wurde, in die Queue eingefügt wird, wird $\texttt{parent}[i] = u$ gesetzt.
   Der Pfad von s nach t ergibt sich dann aus dem Array $\langle t, u = \texttt{parent}[t], v = \texttt{parent}[u], \ldots, s = \texttt{parent}[\cdot]\rangle$ in umgekehrter Reihenfolge. Dieses Array kann durch iteratives Auslesen des `parent`-Arrays konstruiert werden. Es können maximal $n-2$ Knoten auf dem Pfad von s nach t liegen, also benötigt das Bestimmen dieses Pfades Zeit in $O(n)$.

3. Idee: Ist ein Graph bipartit, können wir seine Knoten so mit zwei Farben färben, dass für einen Knoten $v \in V$ alle Knoten in $N(v)$ eine andere Farbe haben als v.

```
1:  BIPARTITEBFS(G = (\{v_0, ..., v_{n-1}\}, E) : Graph) : Bool
2:      color : [ℕ ∪ {⊥}; n] = ⟨⊥ ..., ⊥⟩
3:      Q: Queue
4:      Q.PUSH(v_0)
5:      color[0] = 0
6:      while ¬Q.empty do
7:          v_i: Node = Q.POP()
8:          for v_k ∈ N(v_i) do
9:              if color[k] = ⊥ then                    // v_k hat noch keine Farbe
10:                 Q.PUSH(v_k)
11:                 color[k] = (color[i] + 1) mod 2
12:             else if color[k] = color[i] then        // ungerader Kreis gefunden
13:                 return false
14:             end
15:         end
16:     end
17:     return true
```

Dieser Algorithmus bestimmt, ob seine Eingabe bipartit ist. Zwei benachbarte Knoten können nicht in der gleichen Partition liegen, denn zwischen ihnen existiert eine Kante. Gelingt es uns andererseits, die Knoten so in zwei Mengen zu unterteilen, dass keine zwei benachbarten Knoten in der gleichen Menge liegen, wissen wir, dass die Graph bipartit ist. Der Algorithmus bestimmt solche Mengen, indem er jedem Knoten eine von zwei möglichen Farben zuweist. Hat ein Knoten $v \in V$ eine Farbe zugewiesen bekommen, müssen also alle Knoten aus $N(v)$ die andere erhalten können.

Wahlen der Verfassten Studierendenschaft

Du möchtest gerne mitbestimmen, wer uns Studierende gegenüber dem KIT vertritt?
Vom **30. Mai bis 03. Juni** wird gewählt!
Dabei geht es um den **Fachschaftsvorstand**, der die Studierenden gegenüber der Fakultät vertritt und die Fachschaftsarbeit koordiniert, und um das **Studierendenparlament**, welches das KIT-weite legislative Gremium der VS ist.
Gewählt werden kann den ganzen Tag über bei den Wahlurnen, insbesondere im Mathebau-Atrium, vor dem Infobau und in der Mensa. Außerdem gibt es dort **Kekse**!