Lehrstuhl Algorithmen
und Komplexität
Prof. Dr. Fabian Kuhn
Sebastian Daum, Oleksii Saukh

Algorithmen und Datenstrukturen
(Informatik II) SS 2014

http://ac.informatik.uni-freiburg.de/teaching

UNI FREIBURG

Übungsblatt 8

Abgabe: Dienstag, 8. Juli, 16:00 Uhr

Laden Sie Ihre Lösungen für theoretische *Übungen im Format "loesung-xx-y.pdf" hoch, wobei "xx" die Nummer des aktuellen Übungsblattes ist (bei Bedarf mit führender Null) und "y" die Aufgabennummer, d.h., erzeugen Sie* pro (theoretische) Aufgabe *eine* .pdf*-Datei. Scans, Bilder von Scans etc. bitte zuerst ins pdf-Format umwandeln. Bei Programmieraufgaben müssen Sie – wann immer es relevant/durchführbar ist – Unit Tests sowie einen Style-check durchführen. Laden Sie außerdem eine Datei "erfahrungen.txt" hoch, in welcher Sie Ihre Erfahrungen und Meinungen zu dem Übungsblatt teilen.*

Aufgabe 1 (5 Punkte)

Gegeben sei ein *gerichteter* Graph G mit n Knoten $v_1, \dots, v_n$, welcher in Form einer $n \times n$ Adjazenzmatrix $A = (a_{i,j})_{i,j \leq n}$ abgespeichert ist. Gibt es eine Kante von v_i nach v_j, dann ist in $a_{i,j} = 1$, ansonsten 0. In dem Graphen sollen Sie nun einen Knoten v_s finden, mit folgenden Eigenschaften:

- *Jeder* andere Knoten $v \in V \setminus \{v_s\}$ hat eine gerichtete Kante auf v_s.
- v_s hat keine einzige ausgehende Kante.

Geben Sie einen Algorithmus an (idealerweise in Pseudo-Code), welcher einen solchen Knoten in Laufzeit $O(n)$ findet (falls es keinen Knoten geben sollte, soll der Algorithmus dies erkennen und eine Fehlermeldung zur—ückgeben. Begründen Sie kurz, warum Ihr Algorithmus funktioniert.

Hinweis: Es hilft, sich zuerst einmal die Aufgabenstellung unter der Annahme zu betrachten, dass im Voraus bekannt ist, dass exakt ein solcher Knoten v_s im Graphen existiert.

Aufgabe 2 (5 Punkte)

Ein (ungerichteter) Multigraph G ist ein Graph, in welchem zwischen zwei Knoten u und v mehrere identische Kanten $\{u, v\}$ existieren können; d.h., in der Adjazenzliste des Knotens u taucht der Zeiger auf den Knoten v schlicht mehrfach auf. E ist hierbei eine Multimenge, das Element $\{u, v\}$ kann mehrfach in ihr vorkommen, z.b., $E = \{\{1, 2\}, \{1, 2\}, \{1, 3\}\}$ (die Mächtigkeit von E wäre in diesem Fall $|E| = 3$).

Sei nun G ein Multigraph mit $V = \{1, 2, \dots, n\}$ und $m = |E|$ Kanten, abgespeichert als Adjazenzlisten mit zugehörigem Knoten-Array A. Knoten 1 ist in $A[1]$ zu finden, sowie ein Zeiger auf den Anfang seiner Adjazenzliste, Knoten 2 in $A[2]$, etc.

Finden Sie einen Algorithmus, der diesen Multigraph in einen normalen Graphen $G' = (V, E')$ transformiert, in welchem alle mehrfach vorkommenden Kanten durch eine einzelne ersetzt werden. Um die volle Punktzahl zu erhalten, sollte Ihr Algorithmus nur $O(|V|+|E|)$ Laufzeit und $O(|V|+|E|)$ Speicher benötigen. Eine Begründung Ihrer Lösung ist nicht notwendig.

Bemerkung: Für das Lösen der Aufgabe ist diese Bemerkung zwar irrelevant, aber in so einem Graphen muss nicht $m \in O(n^2)$ gelten.

Aufgabe 3 (5 Punkte)

Gegeben sei ein *gerichteter* Graph $G = (V, E)$, abgespeichert via Adjazenzlisten, und ein Knoten $v \in V$. Geben Sie einen Algorithmus an, mit welchem Sie in *möglichst kurzer Zeit* die Menge $U = \{u \in V : \exists$ Pfad von u nach $v\}$

finden, d.h., alle Knoten u von welchen aus ein Pfad nach v existiert.

Aufgabe 4 (5 Punkte)

Gegeben ist ein *gerichteter* Graph $G = (V, E)$, eine Gewichtsfunktion $w : E \to \{1, 2\}$[1], sowie zwei Knoten s und t aus V. Der gewichtete gerichtete Graph G ist dabei via Adjazenzlisten gespeichert. Beschreiben Sie einen Algorithmus, welcher den *kürzesten* (d.h., leichtesten) Pfad in G von s nach t findet, und das in $O(n + m)$ Zeit. Begründen Sie kurz, warum ihr Algorithmus funktioniert, sowie seine Laufzeit.

[1] Ein gewichteter Graph wird mathematisch normalerweise als Graph und einer Funktion $w : E \to W$ dargestellt (W ist dabei eine Menge für Gewichte), weil w so jeder Kante eindeutig genau ein Gewicht zuordnet. In der Informatik wird das Gewicht einer Kante einfach als zusätzliche Information in jedem Element einer Adjazenzliste gespeichert, oder, im Fall einer Adjazenzmatrix, als Wert in der Matrix (statt 0 und 1).