Albert-Ludwigs-Universität, Inst. für Informatik
Prof. Dr. Fabian Kuhn
Y. Maus, O. Saukh 10 Juni, 2016

Algorithmen und Datenstrukturen
Sommersemester 2016
Übungsblatt 7

Abgabe bis 12:00, Freitag, 17 Juni, 2016

Achtung: Bitte fügen Sie Ihrer Lösung eine (kurze) Datei *erfahrungen.txt* mit Ihren Erfahrungen mit dem jeweiligen Aufgabenblatt hinzu. Das Forum können (und sollen) Sie nicht nur für Fragen zur Übung, sondern auch für Fragen zur Vorlesung benutzen.

Aufgabe 1 (6 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 $\mathcal{O}(n)$ findet. Falls es keinen Knoten geben sollte, soll der Algorithmus dies erkennen und eine Fehlermeldung zurückgeben. 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 (6 Punkte)

Zum topologischen Sortieren eines zyklenfreien gerichteten Graphen $G = (V, E)$ kann man eine DFS-Traversierung von einem beliebigem Knoten starten. Dann liefert die **umgekehrte** Besuchsreihenfolge der Traversierung eine topologische Sortierung von G.
Sei nun $G' = (V, E')$ der Graph, der sich durch Umdrehen von jeder Kante aus G ergibt. Zeigen Sie, dass die Besuchsreihenfolge (**nicht umgekehrt**) einer beliebigen DFS-Traversierung von G' auch eine topologische Sortierung von G liefert.

Aufgabe 3: BFS vs DFS (8 Punkte)

In dieser Aufgabe sollen Sie in einem ersten Schritt eine gegebene Klasse, welche gerichtete Graphen darstellt, um Navigationsfunktionen erweitern und diese dann auf einem Graph, der auf dem Freiburger Verkehrsnetz basiert, anwenden.

- (4 Punkte) Im *public* SVN Verzeichnis finden Sie eine Klasse *Graph*, welche gerichtete Graphen implementiert. Erweitern Sie diese Klasse um die Methoden *bfs_path(src, dst)* und *dfs_path(src, dst)*. Beide Methoden erhalten zwei Knoten IDs (*src* und *dst*) als Argumente und sollen einen Pfad von *src* nach *dst* durch eine Abwandlung der Standard BFS bzw. DFS Traversierung berechnen, d.h. die Traversierung beginnt bei *src* und wird abgebrochen, sobald der Knoten *dst* gefunden wurde. Beide Funktionen sollen die IDs der besuchten Knoten in der Besuchsreihenfolge zurückgeben.

- (3 Punkte) Im *public* SVN befindet sich eine Datei *map.txt*, die eine Graphdarstellung G des Freiburger Verkehrsnetz beinhaltet. Lesen Sie die Datei ein und erstellen einen Graph mit der vorgegebenen Klasse. Das Dateiformat ist am Ende dieser Übung erklärt.

  Eine zweite Datei *list.txt* enthält eine Liste von Knoten IDs $L = \{v_1, v_2, \ldots, v_n\}$ von G.

  Sie sollen nun mit Hilfe der Funktionen *bfs_path(src, dst)* und *dfs_path(src, dst)* zwei Navigationstouren von v_1 nach v_n berechnen - dabei sollen jedoch auch die Zwischenziele $v_2, \ldots, v_{n-1}$ in der vorgegebenen Reihenfolge besucht werden und es soll mit der jeweiligen Methode von v_i zu v_{i+1} navigiert werden.

  Exportieren Sie die GPS Koordinaten der Touren in die Dateien *bfs.csv* und *dfs.csv*.

- (1 Punkt) Laden Sie beide Dateien (*bfs.csv* und *dfs.csv*) auf die Web-Seite

  `http://www.gpsvisualizer.com`

  hoch um die gefundend Pfade zu visualisieren. Fügen Sie die Visualisierung Ihrer Lösung hinzu (ein Bildschirmfoto ist ausreichend).

  Überlegen Sie sich in wiefern die beiden Funktionen tatsächlich zur Navigation genutzt werden können und erklären Sie kurz ihre Vermutungen in der Datei *erfahrungen.txt*.

Hinweis: Sie können gerne weitere Funktionen implementieren und versuchen selbst Eigenschaften des Verkehrsnetzes heraus zu finden.

Datei Formate

map.txt

Zeile 0: *int* N - Anzahl der Knoten im Graph
Zeilen $1 \ldots N$: Knoten. Erste Zahl ist *int* ID, zweite Zahl *float* Breitengrad des Knotens, dritte Zahl *float* Längengrad des Knotens. Die Zahlen sind durch Leerzeichen getrennt.
Zeile $N + 1$: *int* M - Anzahl der Kanten im Graph
Zeilen $N + 2 \ldots N + M + 1$ - Kanten. Die ersten zwei Zahlen sind die *int* IDs von inzidenten Knoten. Die dritte Zahl ist das *float* Gewicht der Kante. Die Zahlen sind durch Leerzeichen getrennt.

list.txt

Zeile 0: *int* N - Anzahl der Knoten IDs
Zeilen $1 \ldots N$ - *int* Knoten IDs.

dfs.csv, bfs.csv

Jede Zeile muss den Breitengrad und Längengrad als *float* und durch ein Komma getrennt enthalten.