Universität Stuttgart

Übungsblatt 7

Datenstrukturen und Algorithmen (SS 2022)

Abgabe: Montag, 13.06.2022, 15:30 Uhr — Besprechung: ab Montag, 20.06.2022

Abgabevorschriften: Die Aufgaben auf diesem Blatt sind unter Einhaltung der Abgabevorschriften[1] zu lösen und abzugeben.

Lernziele: Nach dem Tutorium zu diesem Blatt sollten Sie folgende Lernziele erreicht haben. Wenn nicht, zögern Sie nicht, Ihre:n Tutor:in anzusprechen um die Lücken zu füllen, Unklarheiten zu klären oder Fragen zu beantworten.

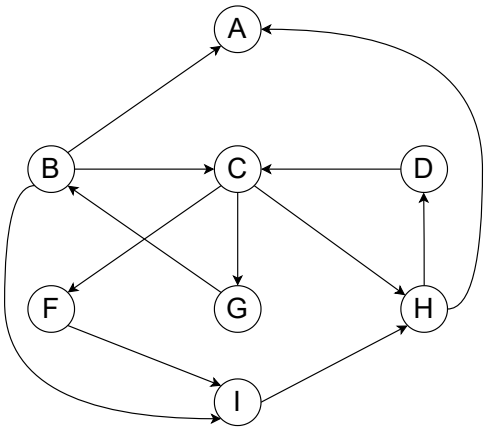
- Sie kennen die vorgestellten Graph-Traversierungen Breitensuche, Tiefensuche und topologische Sortierung und können sie auf einem gegebenen Graphen durchführen.
- Sie wissen wie der Dijkstra-, A*- und Bellman-Ford-Algorithmus funktionieren, welche Eigenschaften sie haben und können sie in Java implementieren.
- Sie wissen was ein minimaler Spannbaum ist und welche Eigenschaften er hat.
- Sie können den Algorithmus von Kruskal anwenden, um einen minimalen Spannbaum eines zusammenhängenenden Graphen zu konstruieren.

Punkte: Dieses Übungsblatt beinhaltet 6 Aufgaben mit einer Gesamtzahl von 30 Punkten. Zum Bestehen werden also 15 Punkte benötigt.

Aufgabe 1 Breitensuche und Tiefensuche verstehen *[Punkte: 5]*

In dieser Aufgabe werden Sie sowohl Breiten- als auch Tiefensuche durchführen und vertiefen.

Gegeben sei hierfür der folgende Graph G_1:



Hinweis: Im Anhang finden Sie Graph G_1 mehrfach abgebildet. Diese Hilfsseite können Sie verwenden, um die Traversierungen schrittweise durchzuführen. Sie benötigen gegebenenfalls mehrere Kopien der Hilfsseite, um Breiten- und/ oder Tiefensuche schrittweise zu visualisieren.

[1] https://ilias3.uni-stuttgart.de/goto_Uni_Stuttgart_file_2904210.html

(a) In der Vorlesung haben Sie die *Breitensuche* kennengelernt.

i. *(1 Punkt)* Geben Sie die Reihenfolge an, in der die Knoten bei der *Breitensuche* von Graph G_1 ausgehend von Knoten B schwarz gefärbt werden.

ii. *(1 Punkt)* Geben Sie die Reihenfolge an, in der die Knoten bei der *Breitensuche* von Graph G_1 ausgehend von Knoten H schwarz gefärbt werden.

Konvention: Wenn ein Knoten mehrere adjazente Knoten hat, arbeiten Sie diese in lexikographischer Reihenfolge, d.h. A < B < C < ..., ab.

(b) In der Vorlesung haben Sie die *Tiefensuche* kennengelernt.

i. *(1 Punkt)* Geben Sie die Reihenfolge an, in der die Knoten bei der *Tiefensuche* von Graph G_1 ausgehend von Knoten I schwarz gefärbt werden.

ii. *(1 Punkt)* Geben Sie die Reihenfolge an, in der die Knoten bei der *Tiefensuche* von Graph G_1 ausgehend von Knoten C schwarz gefärbt werden.

Konvention: Wenn ein Knoten mehrere adjazente Knoten hat, arbeiten Sie diese in lexikographischer Reihenfolge, d.h. A < B < C < ..., ab.

(c) *(1 Punkt)* Auf Blatt 6 haben Sie sich mit *Zusammenhängen* in Graphen auseinandergesetzt. Geben Sie mit Hilfe der *Breitensuche* oder *Tiefensuche* einen Algorithmus in Pseudocode-Schreibweise an, der alle *maximalen Zusammenhangskomponenten* eines *ungerichteten* Graphen G bestimmt.

Aufgabe 2 Topologische Sortierung *[Punkte: 5]*

(a) *(1 Punkt)* Erläutern Sie in eigenen Worten, was eine *topologische Sortierung* auf einem Graphen ist.

(b) *(1 Punkt)* Jede *topologische Sortierung* ist eindeutig. Bedeutet das auch, dass es für jeden beliebigen Graphen maximal eine *topologische Sortierung* existiert? Begründen Sie Ihre Antwort. Sie können Ihre Begründung durch einen Beispielgraphen illustrieren. Eine Antwort ohne Begründung wird mit null Punkten bewertet.

(c) *(2 Punkte)* Gegeben sei ein gerichteter Graph $G_2 = (V, E)$ mit $V = \{A, B, C, D, E, F, G\}$ und $E = \{(A, B), (A, D), (C, B), (D, B), (D, F), (E, A), (F, B), (G, E)\}$. Stellen Sie diesen in Form eines *Knoten-Kanten-Diagramms* dar. Bestimmen Sie dann eine korrekte *topologische Sortierung*. Dokumentieren Sie Ihr Vorgehen schrittweise.

(d) *(1 Punkt)* Wie würde die *topologische Sortierung* aussehen, wenn Sie eine Kante von D zu G zeichnen würden? Begründen Sie Ihre Antwort. Eine Antwort ohne Begründung wird mit null Punkten bewertet.

Aufgabe 3 Impl Dijkstra implementieren *[Punkte: 8]*

Ziel dieser Aufgabe ist es, den kürzeste Wege in der Wilhelma zu finden. Hierfür soll der Dijkstra-Algorithmus verwendet werden, den Sie in der Vorlesung kennengelernt haben. Hierfür sind im Eclipse-Projekt folgende Codefragmente gegeben:

- Schnittstelle `IEdge` und bereits ausimplementierte Klasse `Edge`
- Schnittstelle `INode` und bereits ausimplementierte Klasse `Node`
- Schnittstelle `IWeightedGraph` und bereits ausimplementierte Klasse `WeightedGraph`
- Schnittstelle `IShortestPath` und unvollständige Klasse `ShortestPath`
- Klasse `Wilhelma`

Vervollständigen Sie die Klasse `ShortestPath`, die `IShortestPath` implementiert.

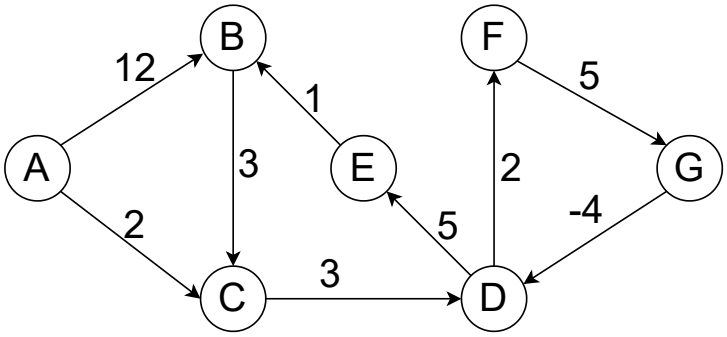
Hinweise:

- Lesen Sie den vorgegebenen Code sorgfältig.
- Modifizieren Sie ausschließlich die zu implementierenden Methoden der Klasse *ShortestPath*. Alle anderen Klassen und Interfaces dürfen **nicht** verändert werden.
- Aufgrund der langen KnotenIDs in den Open-Streetmap-Daten werden zur Indizierung der Knoten 64-bit Integer (`long`) verwendet.
- Die Verwendung der Java Bibliothek java.util.PriorityQueue kann hilfreich sein. Achten Sie dabei darauf, dass Prioritätswerte in PriorityQueues nicht verändert werden können. Stattdessen müssen Sie die zu verändernden Elemente herausnehmen und nach dem Update wieder hinzufügen.
- Wählen Sie geeignete Hilfsstrukturen, z.B. um die Distanz zu den einzelnen Knoten, oder die bereits besuchten Knoten zu speichern.
- Nutzen Sie die bereits existierenden und die von Ihnen zu implementierenden Methoden um den Dijkstra Algorithmus zu implementieren.
- Vorgehen, um den gefundenen kürzesten Weg in der Wilhelma zu visualisieren:
  1. Führen Sie `Wilhelma.java` aus.
  2. Kopieren Sie die den gesamten Inhalt der Konsole.
  3. Öffnen Sie in einem Browser Ihrer Wahl die Webseite `www.geojson.io`.
  4. Fügen Sie den kopierten Inhalt nun in das rechte Feld auf der Webseite ein.
- Zu Beginn wird der kürzeste Weg vom Haupteingang zum Schwingaffenhaus berechnet. Sie können aber auch unterschiedliche Start- und Endpunkte in der Wilhelma ausprobieren. Ersetzen Sie hierfür die Werte für `entry` und `destination` in der Klasse `Wilhelma` durch einen der vorausgewählten, aber auskommentierten Werte. Falls Sie die IDs direkt aus der `Wilhelma.osn` ziehen, achten Sie darauf, dass nicht alle Knoten im Datensatz erreichbar sind, weil manche Wege in Gehegen liegen, die nicht zugänglich sind.

Aufgabe 4 Dijkstra- und Bellman-Ford-Algorithmus verstehen *[Punkte: 5]*

In dieser Aufgabe geht es um zwei der vorgestellten Algorithmen für kürzeste Wege, den Dijkstra- und Bellman-Ford-Algorithmus. Dabei wird in dieser Aufgabe ausschließlich der *einfache Bellman-Ford-Algorithmus* und **nicht** der *erweiterte Bellman-Ford-Algorithmus* betrachtet.

Gegeben sei der Graph G_3:



(a) *(1 Punkt)* Sie haben in der Vorlesung den *Dijkstra-Algorithmus* kennengelernt. Dieser Algorithmus führt angewendet auf Graphen mit negativen Kantengewichten nicht zwangsläufig zu korrekten Ergebnissen. Zeigen Sie dies anhand eines einfachen Beispiels und erläutern Sie den Grund für diese Tatsache anhand Ihres Beispiels.

(b) *(3 Punkte)* Wenden Sie den *Bellman-Ford-Algorithmus*, den Sie in der Vorlesung kennengelernt haben, an, um in G_3 den kürzesten Pfad von A zu allen anderen Knoten zu ermitteln. Stellen Sie Ihr Vorgehen mit Hilfe des Templates in Tabelle 1 dar. Falls Sie dafür nicht alle Zeilen benötigen, streichen Sie die restlichen durch.

Hinweis: Ein Schritt, bzw. eine Zeile, entspricht hierbei einem Durchlauf der äußeren Schleife des Algorithmus.

Schritt	Kosten						
	A	B	C	D	E	F	G
Initialisierung							
1							
2							
3							
4							
5							
6							
7							
8							
9							

Tabelle 1: Template für die Dokumentation des Bellman-Ford-Algorithmus.

(c) *(1 Punkt)* Nehmen Sie an, die Kante von Knoten G zu D in G_2 hat ein Gewicht von -10. Ist der *Bellman-Ford-Algorithmus* immer noch anwendbar? Begründen Sie Ihre Antwort. Eine Antwort ohne Begründung wird mit null Punkten bewertet.

Aufgabe 5 A*-Algorithmus anwenden *[Punkte: 3]*

In dieser Aufgabe sollen Sie mit Hilfe des *A*-Algorithmus* den kürzesten Weg zwischen zwei Punkten auf der Karte in Abbildung 2 finden.

Die Karte besteht aus unterschiedlich gefärbten Feldern. Die hellen Felder stellen zugängliche Bereiche dar, wohingen die grauen Felder für Hindernisse stehen, die nicht betreten werden können. Jedes helle Feld entspricht außerdem einem Knoten in einem Graphen (den Sie mit dem entsprechenden Buchstaben referenzieren können), von dem aus man zu den angrenzenden hellen Feldern bzw. Knoten gelangen kann. Es sind nur vertikale und horizontale Bewegungen zu direkt angrenzenden Feldern möglich, was den Kanten im zugehörigen Graphen entspricht. Das heißt insbesondere diagonale Bewegungen und Sprünge zu entfernteren Feldern sind nicht möglich.

Wenden Sie den *A*-Algorithmus* an, um den kürzesten Weg von A nach G in Abbildung 2 zu finden. Dabei verursachen sowohl horizontale, als auch vertikale Schritte Kosten in Höhe von 1. Um die Entfernung h zwischen zwei Feldern a und b abzuschätzen, verwenden Sie die *Manhattan-Distanz* $d(a, b) = |a_x - b_x| + |a_y - b_y|$. Sollten während der Durchführung des A*-Algorithmus mehrere Knoten den selben f-Wert aufweisen, wählen Sie den Knoten, der in der lexikographischen Reihenfolge entsprechend zuerst ist.

Dokumentieren Sie genau, wie Sie den Algortihmus durchführen und geben Sie für jeden Schritt, d.h. zu jedem Schleifendurchlauf im Pseudocode, mindestens an welchen Knoten Sie betrachten und welchen Inhalt die Queues *OPEN* und *CLOSED* danach haben. Geben Sie schließlich die Länge des kürzesten Weges von A nach G, sowie den konkreten Pfad (d.h., A, ..., G) an. Sie können zur Dokumentation das Template aus Tabelle 5 verwenden. Streichen Sie dann nicht benötigte Zeilen in der Tabelle durch.

A	B				
C	D			E	
F		G			H
I	J	K	L		
M		N			
		O	P	Q	R

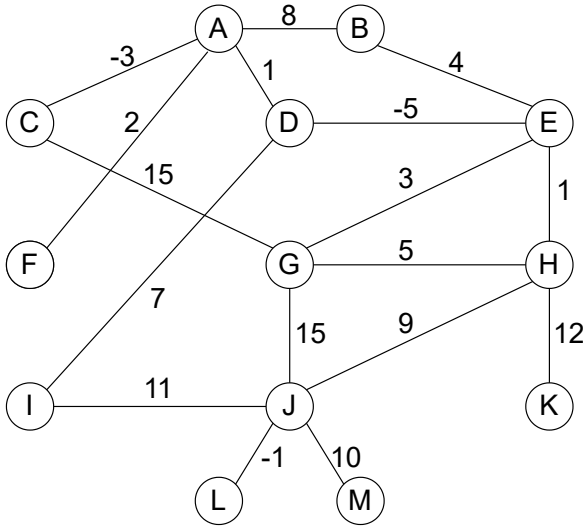
Tabelle 2: Karte für A* Algorithmus.

Schritt	Knoten	OPEN	CLOSED
Initialisierung			
1. Schritt			
2. Schritt			
3. Schritt			
4. Schritt			
5. Schritt			
6. Schritt			
7. Schritt			
8. Schritt			
9. Schritt			
10. Schritt			

Tabelle 3: Template für Dokumentation des A*-Algorithmus.

Aufgabe 6 Minimaler Spannbaum *[Punkte: 4]*

Gegeben sei der folgende *ungerichtete* Graph G_4:



(a) *(2 Punkte)* In der Vorlesung haben Sie den *Algorithmus von Kruskal* kennengelernt, um aus einem Graphen einen *minimalen Spannbaum* zu konstruieren. Wenden Sie diesen auf G_4 an und zeichnen Sie den sich aus G_4 ergebenden minimalen Spannbaum G_5. Erläutern Sie Ihr Vorgehen.

(b) Machen Sie sich mit dem *Traveling Salesman Problem (TSP)* vertraut. Außerdem sei der Algorithmus A_1 gegeben:

Input: Graph G
Output: Integer distance
Integer $distance = 0$;
Bilde den den minimalen Spannbaum G' durch Anwenden des Algorithmus von Kruskal auf G;
Wähle einen beliebigen Startknoten s;
Durchlaufe G' im Sinne des Euler'schen Kreises (d.h. jede Kante wird genau einmal in jede Richtung durchlaufen), inkrementiere *distance* dabei um die jeweiligen Kantengewichte;
return *distance*;

i. *(1 Punkt)* Berechnet A_1 die Lösung *Traveling Salesman Problems* oder stellt es lediglich eine obere odere untere Grenze dar? Begründen Sie Ihre Antwort. Eine Antwort ohne Begründung wird mit null Punkten bewertet.

ii. *(1 Punkt)* Für den Fall, dass A_1 nicht die Lösung, sondern lediglich eine obere oder untere Grenze des *Traveling Salesman Problems* berechnet, gibt es eine Möglichkeit A_1 zu verbessern, um eine bessere Abschätzung der Lösung zu erhalten? Begründen Sie. Für den Fall, dass A_1 die Lösung des *Traveling Salesman Problems* berechnet, führen Sie den Algorithmus anhand eines beliebigen Beispiels durch und erläutern Sie es. Eine Antwort ohne Begründung bzw. Erläuterung wird mit null Punkten bewertet.

Anhang

Graph G_1 für Breiten- und Tiefensuche