Technische Universität Wien
Institut für Computergraphik und Algorithmen
Arbeitsbereich für Algorithmen und Komplexität

186.866 Algorithmen und Datenstrukturen VU

Programmieraufgabe E2

---

1 Übersicht

Ihre Aufgabe ist es die effizientere Variante des Dijkstra Algorithmus aus der Vorlesung, sowie die dazu gehörige Datenstruktur, eine Priority Queue, in Form eines Min-Heaps zu implementieren.

Das benötigte Framework („AlgoDat_E2.zip“) steht im TUWEL zu Verfügung. Implementieren Sie in der Datei „exercise/E2.java“ folgende vorgegebenen Methoden.

2 Dijkstra

Für den Dijkstra-Algorithmus implementieren Sie die Methode *shortestPaths* wobei **G** ein gerichtete Graph ist und **s** und **t** die IDs der Knoten, zwischen denen Sie den **kürzesten Pfad** finden sollen. **Q** ist eine bereits korrekt initialisierte, leere Priority Queue. Als Rückgabe erwartet das Framework dann das Gewicht eines kürzesten Pfades von s nach t. Weiterhin speichern sie bitte In **path** die IDs der besuchten Knoten, auf dem kürzesten Pfad von s nach t.

Der Graph **G** wird als ein Objekt der Klasse *ADGraph* übergeben. Diese Klasse stellt ihnen folgende Methoden zur Verfügung. Im Folgenden bezeichnen wir mit **u** und **v** die Knoten mit den IDs **u** und **v**.

- *outNeighbors(int v)* liefert ihnen eine Liste des Typs *ArrayList<Integer>* zurück, welche die IDs der ausgehenden Nachbarn des Knotens **v** enthält.
- *weight(int u, int v)* liefert ihnen das Gewicht der Kante zwischen **u** und **v** zurück.
- *numVertices()* liefert ihnen die Gesamtanzahl der Knoten im Graphen zurück. Sie können davon ausgehen, dass die Knoten-IDs **von 0 bis n-1** laufen, wobei **n** die Gesamtanzahl der Knoten ist.

Wie in der Vorlesung besprochen benötigt der Dijkstra-Algorithmus eine Datenstruktur um die abzuarbeitenden Knoten zu verwalten. Hier ist dies die Priority-Queue **Q**. Diese Datenstruktur hat die folgenden von außen zugängliche Methoden.

- *add(int id, double key)* akzeptiert eine ID und einen Key und fügt einen Knoten mit dieser ID und diesem Key in den Heap ein.
- *decreaseKey(int id, double key)* akzeptiert die ID eines Knotens, sowie einen neuen Key und reduziert den gespeicherten Key auf den neuen, übergebenen Wert.
- *removeMin()* returniert die ID des Knoten mit dem geringsten Key.
- *isEmpty()* returniert *true* wenn der Heap leer ist, sonst *false*.

Weitere, für ihre Implementierung relevante Methoden des MinHeaps, werden im Abschnitt 3 genauer erklärt.

Die Klasse *MinHeap* enthält eine boolean Variable *IS_IMPLEMENTED*. Solange diese auf *false* gesetzt ist wird der Heap bei der Ausführung nicht berücksichtigt. Sie können also ihre Dijkstra Implementierung testen ohne die Klasse *MinHeap* implementiert zu haben. In diesem Fall wird der Algorithmus nur mit einem Fibaonnacci-Heap und einer normalen Queue ausgeführt, welche bereits implementiert sind. Sobald Sie dieses Flag auf *true* setzen wird *shortestPaths* auch mit einer Instanz ihres Heaps aufgerufen.

3 Min-Heap

Als Datenstruktur um die abzuarbeitenden Knoten zu speichern, wird ein **Min-Heap** verwendet. Die oben erwähnten Methoden der Datenstruktur sind bereits vorhanden, der Rest ist von Ihnen zu implementieren.

Hinweis: Der Heap verwendet, wie in der Vorlesung, eine Liste um die Elemente zu speichern, wobei die Position eines Eltern-Knotens mit der Position eines Nachfolger-Knotens zusammenhängt. Die Liste **beginnt bei 0** und die Berechnung von Eltern-, bzw. Nachfolger-Knoten muss dementsprechend angepasst werden (siehe Folie 83).

Im folgenden beschreiben wir die Methoden, welche von ihnen zu implementieren sind. Dies kann komplett in der der Klasse *MinHeap* in der Datei „exercise/E2.java“ erfolgen. Sehen Sie sich auch die zugehörigen Folien aus der Vorlesung an.

- *getParent(int i)* akzeptiert eine Position an der ein Knoten im Heap gespeichert ist und returniert die Position des Elternknotens.
- *getLeft(int i)/getRight(int i)* akzeptiert eine Position an der ein Knoten im Heap gespeichert ist und returniert die Position des linken/rechten Nachfolgers.
- *heapifyUp/heapifyDown* akzeptiert die Position an der ein Knoten im Heap gespeichert ist und verschiebt diesen solange, bis die Heap-Bedingung wieder erfüllt ist. Dies kann nach dem Einfügen, Löschen oder Veränderung eines Keys notwendig sein.

Die folgenden Methoden sind bereits vorhanden und sollten von ihnen bei der Implementierung verwendet werden. Alle folgenden Methoden befinden sich in der Datei "PartialMinHeap". Dies ist die Super-Klasse von "MinHeap". "MinHeap" hat damit Zugriff auf beide aufgeführten Methoden.

- *swap(int i, int j)* akzeptiert die Position zweier Elemente im Heap und vertauscht ihre Positionen im Heap.
- *keyAt(int i)* akzeptiert eine Position im Heap und returniert den Key des Knotens an dieser Stelle.

Wenn sie ihren Dijkstra implementiert haben und das Programm ausführen (siehe „Tutorial.pdf“) können sie "Vienna.csv" oder "Small.csv" auswählen um ihre Implementierung an echten Straßendaten bzw. einem kleinen Graphen zu testen. Auf der Instanz "Vienna.csv" werden 50 Routen berechnet. Die Instanz "Small.csv" ist dem Wikipedia-Artikel über den Dijkstra-Algorithmus entnommen, in dem sie auch eine Referenzlösung finden. Beachten sie dass die Laufzeiten bei der kleinen Instanz nicht aussagekräftig sind. Diese dient hauptsächlich als Testinstanz für ihre Dijkstra Implementierung.

4 Auswertung

Wenn sie die Datei "AlgoDat_E2.java" ausführen können sie, wie im Tutorial beschrieben, in der Konsole eine Instanz auswählen. Es wird eine *.csv-Datei* im *solutions* Ordner erstellt, welche sie mit der Datei "E2/Evaluation/plot.html" auswerten können. Sie bekommen dann einen Vergleich der Laufzeiten relativ zur Pfadlänge und für alle verwendeten Datenstrukturen präsentiert.

Beachten sie dass die Laufzeiten bei der kleinen Instanz nicht aussagekräftig sind. Vergleichen sie die Laufzeiten der Datenstrukturen und beantworten sie folgende Fragen:

- **Welche Unterschiede** sehen sie zwischen der Implementierung mit einem Heap und einer Liste? Begründen Sie ihre Antwort.
- Sehen sie einen Unterschied zwischen den Laufzeiten **relativ zur Länge der berechneten Route**? Begründen sie ihre Beobachtung.
- **Exportieren sie die Laufzeittabelle**, wie im Tutorial beschrieben, und fügen sie sie in ihre Abgabe ein.

Es wird ihnen weiter eine Karte angezeigt (nur für "Vienna.csv") auf der die berechnet Route angezeigt wird, wobei die blaue Route die Distanz und die grüne Route die Zeit als Kantengewicht verwendet. Diese Karte dient nur der Visualisierung und **muss nicht in die Abgabe eingefügt werden**.

Hinweis: Bevor Sie beginnen, lesen Sie sich unbedingt das Tutorial zu den Programmieraufgaben durch („Tutorial.pdf"). Dieses finden Sie im ZIP-Archiv des Programmier-Frameworks im TUWEL. Diese bietet eine kurze Einführung in das verwendete Framework und erklärt Ihnen detailliert was Sie zu tun haben.