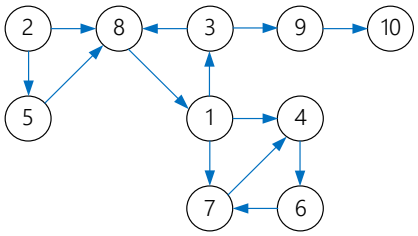
Graphen

Aufgabe 1: Manuelle Graphen-Analyse

Gegeben ist folgender Graph:



Führen Sie für den Graphen G die Breitensuche durch, beginnend im Knoten 1. Geben Sie die Reihenfolge der Knotenbesuche an. «Besuchen» Sie dabei für einen Knoten seine Nachbarn immer in aufsteigender Reihenfolge (also von Konten 1 aus zunächst 3, dann 4, dann 7).

Die Aufgabe muss nicht abgegeben werden.

Aufgabe 2: Graph-Repräsentation

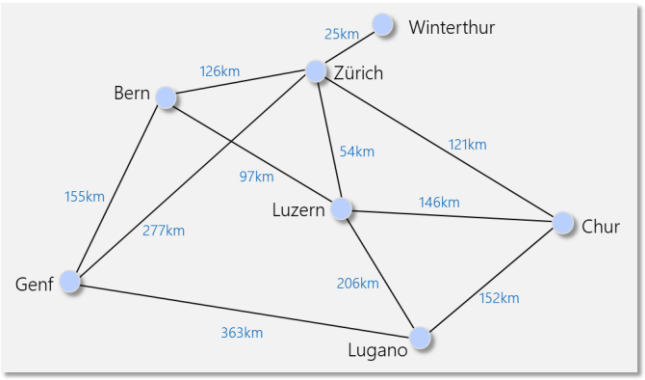
Graphen kann man als Adjazenz-Matrix oder mit Hilfe von Adjazen-Listen darstellen. In dieser Aufgabe untersuchen Sie, welche Darstellungsform für welche Anwendung sinnvoll ist.

a) Geben Sie mindestens drei verschiedene Operationen (wie z.B. einfügen eines Knoten, Löschen einer Kante etc.) an, die in einer Adjazienzmatrix schneller sind als in Adjazenz-Listen.
b) Geben Sie umgekehrt mindestens drei Operationen an, die mit Adjazenz-Listen schneller sind.

Die Aufgabe muss nicht abgegeben werden.

Aufgabe 3: Erzeugen des Graphen

Lesen Sie die Distanzen zwischen den Städten aus der Datei Swiss.txt ein und erstellen Sie einen entsprechenden ungerichteten Graphen. Für diesen Zweck soll eine Klasse RouteServer erstellt werden, die den CommandExecutor implementiert – wie gehabt.



Hinweise:

- Sie können das Interface Graph bzw. die Klasse AdjListGraph verwenden.
- Da der Knoten, die für den Algorithmus notwendigen Felder (dist, mark und prev) nicht enthält, müssen Sie den DijkstraNode verwenden.
- Definition des Graphen: Graph<DijkstraNode,Edge> graph = new AdjListGraph<DijkstraNode, Edge> (DijkstraNode.class,Edge.class);
- addEdge Methode des Graphen verwenden.
- Eine ungerichtete Kante erhalten Sie, indem Sie gerichtete in beide Richtungen einfügen.

Aufgabe 4: Kürzeste Strecke

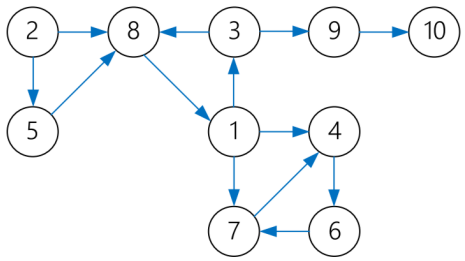
Traversieren Sie den Graphen und bestimmen Sie die kürzeste Verbindung nach dem in Folien beschriebenen Dijkstra-Algorithmus. Geben Sie die gefundene Strecke aus.

Hinweis:
Den Algorithmus (in Pseudocode) aus den Folien verwenden.

- Die Pseudo-Anweisung for all kann einfach in die entsprechende Java 5 Anweisung übersetzt werden.
- Mit der Graph-Methode findNode("Winterthur") kann der Start-Knoten gefunden werden.
- Als Queue nehmen Sie die JDK Queue java.util.Queue<DijkstraNode> pq = new java.util.PriorityQueue<DijkstraNode>(); (Hinweis: "The head of this queue is the least element with respect to the specified ordering".)
- Die Ordnung/Priorität wird durch das Comparable bzw. compareTo bestimmt.
- Vom Zielknoten (Lugano) mittels den gesetzten prev-Verweisen zurück bis nach Winterthur.

Aufgabe 1: Manuelle Graphen-Analyse

Gegeben ist folgender Graph:



Führen Sie für den Graphen G die Breitensuche durch, beginnend im Knoten 1. Geben Sie die Reihenfolge der Knotenbesuche an. «Besuchen» Sie dabei für einen Knoten seine Nachbarn immer in aufsteigender Reihenfolge (also von Konten 1 aus zunächst 3, dann 4, dann 7).

Die Aufgabe muss nicht abgegeben werden.

1 => 3 => 4 => 7 => 8 => 9 => 10

Knoten 2 und 5
werden nie erreicht,
wenn man mit 1
started

Aufgabe 2: Graph-Repräsentation

Graphen kann man als Adjazenz-Matrix oder mit Hilfe von Adjazen-Listen darstellen. In dieser Aufgabe untersuchen Sie, welche Darstellungsform für welche Anwendung sinnvoll ist.

a) Geben Sie mindestens drei verschiedene Operationen (wie z.B. einfügen eines Knoten, Löschen einer Kante etc.) an, die in einer Adjazienzmatrix schneller sind als in Adjazenz-Listen.
b) Geben Sie umgekehrt mindestens drei Operationen an, die mit Adjazenz-Listen schneller sind.

Die Aufgabe muss nicht abgegeben werden.

a) Vorteile für Adjazen-Matrix
- Überprüfen, ob eine Kante zwischen zwei Knoten existiert.

- Bestimmten der Anzahl der ausgehenden Kanten für einen Knoten.

- Überprüfen, ob ein Konten im Graphen existiert.

b) Vorteile für Adjazen-Listen
- Hinzufügen und Löschen einer Kante

- Bestimmen der Nachbarn eines Knoten

- Speicherplatzeffizienz für dünn besetzte Graphen