Ludwig-Maximilians-Universität München
Institut für Informatik
PD Dr. Matthias Renz
Dr. Erich Schubert, Daniel Kaltenthaler

München, 6.7.2015

Algorithmen und Datenstrukturen

SS 2015

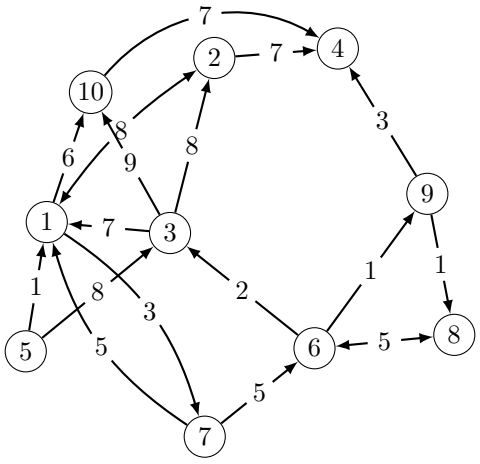
Übungsblatt 11: Graphalgorithmen

Besprechung: 13.7.– 17.7.

Abgabe aller mit **Punkten** versehenen Aufgaben bis 12.7.2015 17:35

Aufgabe 11-1 Graphalgorithmen — Tutoraufgabe

Gegeben ist folgender Graph:



(a) Erstellen Sie die Adjazenzmatrix und Adjazenzlisten des Graphen.
Auf der Diagonalen und statt ∞ dürfen Sie auch $-$ schreiben.

(b) Führen Sie, ausgehend vom Knoten **2** einen Breitendurchlauf und einen Tiefendurchlauf durch den Graphen durch. Zeichnen Sie jeweils den Ergebnisspannbaum, der sich aus Breiten- und Tiefendurchlauf ergibt. Erstellen Sie den Spannbaum so, dass bei Mehrdeutigkeiten zuerst der Knoten mit der kleinsten Knoten-ID besucht wird.

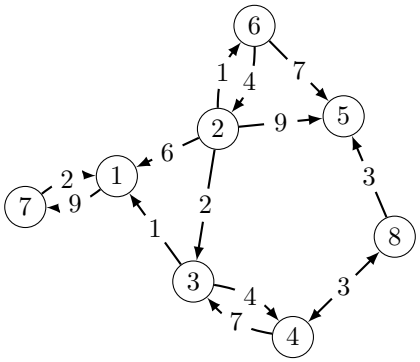
(c) Berechnen Sie die kürzesten Wege ausgehend von Knoten **5** zu allen anderen Knoten. Verwenden Sie dazu den Algorithmus von Dijkstra. Zeichnen Sie für jeden Schritt einen Baum, der nur die kürzesten Wege enthält. Kennzeichnen Sie besuchte Knoten, um sie von "offenen" Knoten zu unterscheiden.

(d) Wenden Sie den Algorithmus von Floyd auf den Graphen an. Füllen Sie entsprechend dem Algorithmus die Kostenmatrix A. Die Einträge von A sollen dabei nicht nur die Kosten, sondern zusätzlich (in Klammern) den Knoten angeben, über den der kostenminimale Pfad läuft. Die Kostenmatrix und die *pathCost*-Matrix werden somit in *einer* Tabelle dargestellt. (Hinweis: der in Klammern angegebene Knoten ist nicht notwendigerweise der nächste Schritt, sondern i.d.R. der letzte Knoten, eine Verbesserung des Weges verursacht hat. Es ist in dem Sinne nur ein Wegpunkt auf einem kürzesten Weg.)

(e) Betrachten Sie den Graphen nun als ungerichteten Graphen. Konstruieren Sie mit Hilfe des Algorithmus von Kruskal den minimalen Spannbaum. Entscheiden Sie, ob das Ergebnis des Algorithmus eindeutig ist und begründen Sie Ihre Antwort.

Aufgabe 11-2 Graphalgorithmen **20 Punkte**

Gegeben ist folgender Graph:



(a) Erstellen Sie die Adjazenzmatrix und Adjazenzlisten des Graphen.
Auf der Diagonalen und statt ∞ dürfen Sie auch $-$ schreiben.

(b) Führen Sie, ausgehend vom Knoten **6** einen Breitendurchlauf und einen Tiefendurchlauf durch den Graphen durch. Zeichnen Sie jeweils den Ergebnisspannbaum, der sich aus Breiten- und Tiefendurchlauf ergibt. Erstellen Sie den Spannbaum so, dass bei Mehrdeutigkeiten zuerst der Knoten mit der kleinsten Knoten-ID besucht wird.

(c) Berechnen Sie die kürzesten Wege ausgehend von Knoten **2** zu allen anderen Knoten. Verwenden Sie dazu den Algorithmus von Dijkstra. Zeichnen Sie für jeden Schritt einen Baum, der nur die kürzesten Wege enthält. Kennzeichnen Sie besuchte Knoten, um sie von "offenen" Knoten zu unterscheiden.

(d) Wenden Sie den Algorithmus von Floyd auf den Graphen an. Füllen Sie entsprechend dem Algorithmus die Kostenmatrix A. Die Einträge von A sollen dabei nicht nur die Kosten, sondern zusätzlich (in Klammern) den Knoten angeben, über den der kostenminimale Pfad läuft. Die Kostenmatrix und die *pathCost*-Matrix werden somit in *einer* Tabelle dargestellt. (Hinweis: der in Klammern angegebene Knoten ist nicht notwendigerweise der nächste Schritt, sondern i.d.R. der letzte Knoten, eine Verbesserung des Weges verursacht hat. Es ist in dem Sinne nur ein Wegpunkt auf einem kürzesten Weg.)

(e) Betrachten Sie den Graphen nun als ungerichteten Graphen. Konstruieren Sie mit Hilfe des Algorithmus von Kruskal den minimalen Spannbaum. Entscheiden Sie, ob das Ergebnis des Algorithmus eindeutig ist und begründen Sie Ihre Antwort.

Aufgabe 11-3 Dijkstra-Algorithmus **6 Punkte**

Gegeben Sei folgende (unvollständige) Implementierung des Dijkstra-Algorithmus auf einer Distanzmatrix. Sie sollen diese Klasse um die fehlende Funktionalität ergänzen.

Beachten Sie, dass diese Implementierung von dem in der Vorlesung vorgestellten abstrakten Graphen abweicht, um den Code für die Übung einfacher zu halten. Transferieren Sie die algorithmischen Überlegungen auf diese reduzierte Datenstruktur. Um den Code einfach zu halten, sollen Sie keinen Heap verwenden, sondern mit linearer Suche den besten Kandidaten identifizieren.

Die Rückgabe des Baumes erfolgt, indem die Arrays `parent[]` mit der Nummer des Vorgängerknotens und `distance[]` mit der Distanz zum Startknoten gefüllt werden. Der Vorgänger des Startknotens `start` soll hierbei auf den Startknoten selbst (`start`) gesetzt werden, und die Distanz auf `0`, wie bereits in dem Code realisiert.

```
public class Dijkstra {
  /**
   * Dijkstra-Algorithmus auf einer Distanzmatrix.
   *
```

```
   * @param m        Distanzmatrix
   * @param start    Startknoten
   * @param parent   RUECKGABE: Vaterknoten
   * @param distance RUECKGABE: Distance von Startknoten
   */
  public static void dijkstra(double[][] m, final int start,
    /* Rueckgabe ueber: */ int[] parent, double[] distance) {
    final int l = m.length; // Anzahl Knoten.
    java.util.Arrays.fill(parent, -1);
    java.util.Arrays.fill(distance, Double.POSITIVE_INFINITY);
    // Startwert richtig eintragen:
    parent[start] = start;
    distance[start] = 0.;

    // TODO: Datenstruktur fuer besuchte Knoten
    for (int i = 0; i < l; i++) {
      int bestpos = -1;
      double bestdist = Double.POSITIVE_INFINITY;
      for (int j = 0; j < l; j++) {
        // TODO: besten Kandidaten auswaehlen
      }
      if (bestpos < 0) {
        break; // Nicht alle Knoten erreichbar
      }
      // TODO: Als besucht markieren
      for (int j = 0; j < l; j++) {
        // Geschlossene Knoten ueberspringen
        double newdist = // TODO: Neue Distanz berechnen
        // TODO: parent[], distance[] aktualisieren
      }
    }
  }

  // TODO: main() Methode zum ausfuehrlichen Testen der Klasse
}
```

Nicht verbundene Knoten haben die Distanz `Double.POSITIVE_INFINITY`.

Der Wert `parent[i] = -1` soll bedeuten, dass Knoten `i` vom Startknoten nicht erreicht werden kann.

Beachten Sie, dass Sie beim Aufrufen dieser Methode zuvor geeignete `parent[]` und `distance[]` Arrays erzeugen und der Methode `dijkstra` übergeben müssen.

(a) Vervollständigen Sie die mit `TODO` gekennzeichneten Stellen.

(b) Fügen Sie eine `main(String[])` Methode zu Ihrer Klasse hinzu, mit der ihr Programm automatisch getestet werden kann. Diese Tests sollen alle wichtigen Fälle ihres Programmes abdecken.

Tipp: mit `System.out.println(java.util.Arrays.toString(array));` können Sie sich den Inhalt eines Arrays einfach ausgeben lassen.

(c) Testen Sie ihr Programm ausführlich.
Fehler, die Sie aufgrund von zu einfachen Tests übersehen haben, müssen Sie verantworten.

Sie dürfen diese Klasse dazu nur an den vorgesehenen Stellen ändern.

Geben Sie diese Klasse als `Dijkstra.java` ab. Klassen die nicht compilieren, oder nicht wie gefordert bearbeitet wurden, müssen von den Korrektoren mit 0 Punkten bewertet werden.