Algorithmen und Datenstrukturen
Sommersemester 2019

TU Berlin, Fakultät IV
Blankertz/Röhr/Stahl
MAR 4-3 und MAR 5-1

Aufgabenblatt 7

Minimale Spannbäume

Wichtiger Hinweis

Die Abgabe von Aufgabe 1 (Programmieraufgabe) ist **optional**. Wer die Aufgaben zu Übungszwecken machen möchte, kann dies gerne tun. Die Korrekturtests laufen normal durch und zeigen entsprechende Punkte an. Alle bekommen die volle Punktzahl für dieses Blatt gutgeschrieben, auch wenn sie nichts abgeben oder fehlerhafte Lösungen einreichen.
Die Tutoriumsaufgaben sind (wie immer) sehr relevant für die Klausur und sollten daher auf jeden Fall bearbeitet werden. Die Tutorien find ganz normal statt.

Abgabe (bis 18.06.2019 19:59 Uhr)

Die folgenden Dateien werden für die Bepunktung berücksichtigt:

`Blatt07/src/EdgeWeightedGraph.java`	Aufgabe 1.1
`Blatt07/src/Clustering.java`	Aufgabe 1.2 bis 1.5

Als Abgabe wird jeweils nur die letzte Version im git gewertet.

Aufgabe 1: Clustering (optional!)

In dieser Aufgabe werden Sie mithilfe des Prim MST Algorithmus ein Clustering Problem lösen. Ein Cluster ist eine Gruppe von Objekten, die über ähnliche Eigenschaften verfügen. Diese Ähnlichkeit bzw. Unähnlichkeit lässt sich zum Beispiel durch eine Norm ausdrücken. In dieser Aufgabe betrachen wir Punkte im $\mathbb{R}^n$ mit der euklidischen Norm.
Clusteranalyse wird in vielen verschiedenen Bereichen angewendet, wie z.B der Bildverarbeitung oder bei der Klassifizierung von Gendaten. Sie werden sich als Anwendungsbeispiel, neben den von uns erstellten n-dimensionalen Graphen, auch den IRIS Datensatz aus dem UCI Machine Learning Repository anschauen. (`https://archive.ics.uci.edu/ml/index.php`)
Der Algorithmus auf dem diese Aufgabe beruht, folgt diesen Schritten:

1. Das Problem in einen gewichteten Graphen umwandeln. Dabei ist für die Kantengewichte z.B. eine geeignete Norm zu wählen.
2. Einen MST auf dem Graphen berechnen.
3. Der Annahme folgend, dass die Knoten, die zu einem Cluster gehören, mit Kanten verbunden sind, die ein kleines Gewicht haben, werden in dem MST nun eine zu bestimmende Anzahl an Kanten mit zu großen Gewichten entfernt.
4. Durch das Entfernen von Kanten aus dem MST, ist dieser nun nicht mehr zusammenhängend. Die einzelnen Zusammenhangskomponeten sind die gesuchten Cluster.

Für Interessierte haben wir im Git Repository einen Artikel abgelegt, der ebenfalls mit einem MST Algorithmus Cluster bestimmt.

1.1 Graphen einlesen (5 Punkte)

Schauen Sie sich den Konstruktor

```
public EdgeWeightedGraph(In in)
```

der Klasse `EdgeWeightedGraph` genau an, um die Datenstruktur zu verstehen. Jeder Knoten in dem Graphen hat eine Position im $\mathbb{R}^n$, die in der Objektvariablen `coord[][]` gespeichert wird. In diesem Konstruktor wird eine Textdatei eingelesen, in der sowohl der Graph durch seine Kanten definiert wird, als auch die Positionen der Knoten im n-dimensionalen Raum. Die Datei hat das folgende Format:

Input schematisch

```
[number of nodes]
[number of edges]
[dimensions]
[1. edge node1] [1. edge node2] [1. coordinate node1]... [last coordinate node1] [1. coordinate node2]... [last coordinate node2]
[2. edge node1] [2. edge node2] [1. coordinate node1]... [last coordinate node1] [1. coordinate node2]... [last coordinate node2]
.
.
.
[last edge node1] [last edge node2] [1. coordinate node1]... [last coordinate node1] [1. coordinate node2]... [last coordinate node2]
```

Im Konstruktor wird der Graph aufgebaut, die Koordinaten gespeichert und die Gewichte der Kanten über die euklidische Norm definiert: die Kante zwischen x_i und x_j hat das Gewicht $\omega_{ij} = \|x_i - x_j\|_2$

Implementieren Sie die Methoden:

```
public double[][] getCoordinates()
```

und

```
public void setCoordinates(double[][] coord)
```

Dies sind die get- und die set-Methoden für die Objektvariable `coord`.

Hinweise:

- Gehen Sie die ganze Klasse `EdgeWeightedGraph` durch, sodass Sie sie danach verstanden haben.
- Das Dateiformat ist sehr ineffizient, da die Koordinaten jedes Knotens für jede seiner Kanten redundant gespeichert ist. Machen Sie sich darüber keine Gedanken.

1.2 Clustering Konstruktoren (0 Punkte)

Schauen Sie sich die beiden Konstruktoren der Klasse `Clustering` an:

```
public Clustering(EdgeWeightedGraph G)
public Clustering(In in)
```

Für den ersten Konstruktor ist der Graph bereits gegeben. In dem zweiten muss dieser erst erstellt werden. Das Inputfile, welches hier eingelesen wird, hat folgende Form:

Input mit Label schematisch

```
[number of nodes]
[dimensions]
[1. coordinate node 1]... [last coordinate node 1] [LABEL]
[1. coordinate node 2]... [last coordinate node 2] [LABEL]
.
.
.
[1. coordinate node n]... [last coordinate node n] [LABEL]
```

Die Datei definiert also die Positionen von Datenpunkten (z.B. mehr-dimensionalen Messwerten), genauer die Koordinaten der Punkte im $\mathbb{R}^n$. Außerdem ist für jeden Datenpunkt ein Label (als `String`) angegeben, welches ihn einem spezifischen Cluster zuordnet. Diese Cluster werden in der Objektvariable `labeled` gespeichert, damit Sie später vergleichen können, ob der Algorithmus die Cluster richtig zugeordnet hat. In dem Konstruktor wird ein vollständiger Graphen erzeugt, in dem jeder der Knoten mit allen anderen verbunden ist.

Hinweis:

- Schauen Sie sich an, wie die Label gespeichert werden, damit Sie diese Information in Aufgabe 1.5 verwenden können.

1.3 Clusteranalyse 1 (30 Punkte)

Implementieren Sie den in der Einleitung erläuterten Algorithmus in einer ersten Variante. Hier wird die Anzahl der zu erwartenden Cluster mitgegeben:

```
public void findClusters(int numberOfClusters)
```

Dafür müssen Sie sich überlegen, welche und wie viele Kanten aus dem MST entfernt werden müssen. Anschließend müssen die Zusammenhangskomponenten gefunden werden. Dazu ist es sinnvoll z.B. eine Hilfsmethode `connectedComponents` zu schreiben, welche die Zusammenhangskomponenten findet. Sie können hier die Klasse `UF` nutzen, welche den Union Find Algorithmus implementiert.

Hinweise:

- Wenn Sie die Zusammenhangskomponenten mithilfe einer Hilfsmethode finden, können Sie diese Funktion in der nächsten Teilaufgabe wiederverwenden.
- Testen Sie Ihre Methode auch mit der Methode `plotClusters()`. Diese visualisiert die Cluster im 2-dimensionalen Raum.

1.4 Clusteranalyse 2 (35+20 Punkte)

Implentieren Sie eine zweite Variante des Algorithmus, in der die Anzahl der Cluster nicht mitgegeben ist:

```
public void findClusters(double threshold)
```

In dieser Variante untersucht der Algorithmus die Änhlichkeit der Kanten des MST mit dem Variationskoeffizienten, dessen Berechnung Sie in dieser Methode implentieren sollen:

```
public double coefficientOfVariation(List <Edge> part)
```

Der Variationkoeffizient ist folgendermaßen defniniert

$$CV(X) = \frac{\sigma}{\mu} = \frac{\sqrt{\frac{1}{n}\sum_{i=0}^{n} x_i^2 - (\frac{1}{n}\sum_{i=0}^{n} x_i)^2}}{\frac{1}{n}\sum_{i=0}^{n} x_i} \tag{1}$$

wobei X ihre Liste ist. Die Methode gibt Ihnen also einen Variationskoeffizienten für eine Liste zurück, Sie wollen aber entscheiden, ob einzelne Kanten entfernt werden oder nicht. Dazu sollten Sie die Kanten, die in Ihrem MST sind, sortieren und dann mit einer Kante anfangen und immer eine weitere hinzunehmen. Wenn der zurückgegebene Koeffizient dann den `threshold` überschreitet, wird diese Kante entfernt. Der neue Knoten, der dadurch verbunden werden würde, ist den anderen Knoten im Cluster nicht ähnlich genug. Wenn Sie die Kanten entfernt haben, gehen Sie vor, wie in 1.3.

1.5 Validierung (20 Punkte)

In dieser Methode implementieren Sie einen einfachen Weg, um zu überprüfen, wie gut der Algorithmus einen gegebenen Datensatz clustert.

```
public int[] validation()
```

Dabei geben Sie ein Array zurück, das als Dimension die richtige Anzahl an Clustern hat, d.h. die Anzahl an Cluster, die gelabelt ist. Für jedes dieser Cluster werden die Punkte gezählt, die in der Analyse richtig zugeordnet wurden und die Anzahl der richtig zugeordneten Punkte wird dann in das Array geschrieben. Wenn mehr Cluster gefunden werden, als es eigentlich geben sollte, reicht es, nur die Anzahl der Cluster zu betrachten, die es wirklich gibt. Danach brechen Sie die Überprüfung einfach ab.

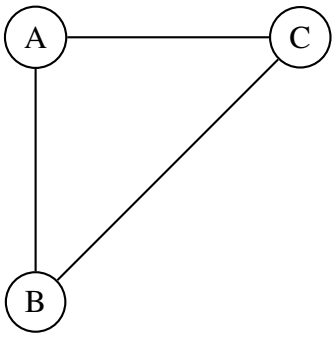
Hinweise:

- Testen Sie diese Methode an dem IRIS Datensatz, den wir Ihnen mitgegeben haben.
- Testen Sie die Methode mit beiden Varianten des Algorithmus (aus Aufgaben 1.3 und 1.4).
- Sie werden feststellen, dass der Algorithmus für die ersten beiden Klassen (= korrekte Cluster) sehr gut funktioniert, für die dritte dagegen nicht.

Aufgabe 2: Union Find (Tut)

2.1 Was ist eine Äquivalenzrelation? Was sind Äquivalenzklassen und wie hängen diese mit Zusammenhangskomponenten zusammen?

2.2 Wozu wird der Union Find Algorithmus benutzt? In welchem Algorithmus spielt er eine Rolle? Wie funktioniert er? Gehen Sie den Algorithmus schrittweise an folgendem Beispiel durch:

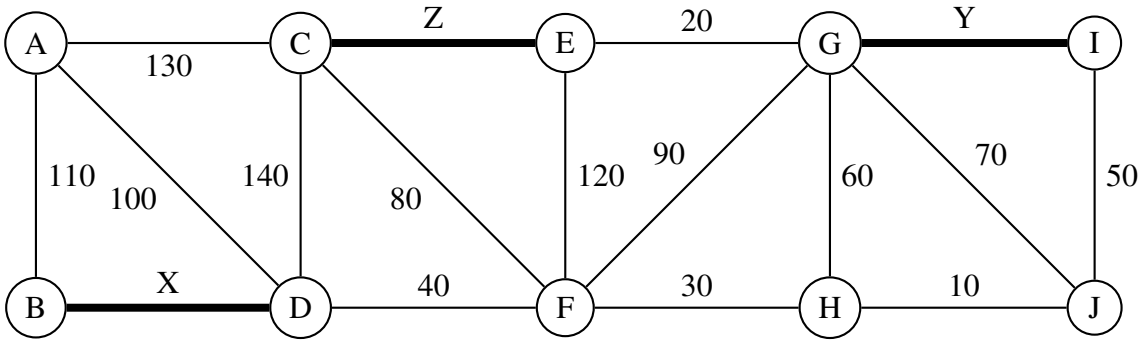


2.3 Funktioniert der folgende Code-Abschnitt für Union Find? Warum wäre das der Fall?

```
public void union(int p, int q) {
   if (connected(p, q)) return;
   for (int i = 0; i < id.length; i++)
      if (id[i] == id[p]) id[i] = id[q];
   count--;
}
```

Aufgabe 3: Minimum Spanning Tree

Nehmen Sie an, der Minimum Spanning Tree (MST) dieses Graphen enthalte die Kanten X, Y und Z.



3.1 Notieren Sie die Gewichte der restlichen Kanten, d.h. alle außer X, Y und Z, die zum MST dieses Graphen gehören müssen unter der obigen Annahme.

3.2 Geben Sie jeweils die größte obere Schranke für die Gewichte der Kanten X, Y und Z an, mit der garantiert ist, das alle drei Kanten tatsächlich Teil des MST sind.
Hinweis: Geben Sie individuell für jedes Gewicht x, y, z eine Schranke an, wobei x, y und z die entsprechenden Gewichte zu den Kanten X,Y und Z sind.

3.3 Setzen Sie für X, Y und Z folgende Gewichte ein und führen Sie Kruskal und Prim ab A aus.

$$x = 120$$
$$y = 50$$
$$z = 80$$