

Übung zur Vorlesung Programmierung

Aufgabe T14

In dieser Aufgabe sollen Sie die Klasse *KMedian* (siehe unten), die eine Implementierung des Algorithmus *k*-Medians bereitstellt, vervollständigen. *k*-Medians ist ein Algorithmus, der beliebige Daten in Untermengen—auch Cluster genannt—unterteilt, sodass sich die Elemente in jedem Cluster möglichst ähnlich sind. Die Ähnlichkeit wird dabei über eine gegebene *Distanzfunktion* gemessen.

Genauer: Der Algorithmus *k*-Medians unterteilt die Menge der Eingabedaten *E* in Teilmengen E_1, \dots, E_k , sodass jedes Element der Menge *E* in genau einer Menge $E_i, i \in \{1, \dots, k\}$ enthalten ist. Dazu geht der Algorithmus wie folgt vor:

1. Wähle zufällig *k* unterschiedliche Elemente e_1, \dots, e_k der Menge *E* als *Mittelpunkte* der Mengen E_1, \dots, E_k .
2. Ordne jedes Element $e \in E$ jener Menge E_i zu, deren Mittelpunkt e bezüglich der verwendeten Distanzfunktion am nächsten ist.
3. Lege für jede Menge E_i den *Median* von E_i als neuen Mittelpunkt e_i fest.
4. Falls das Abbruchkriterium erfüllt ist, gebe $\{(e_1, E_1), \dots, (e_k, E_k)\}$ zurück, ansonsten fahre mit Schritt 2 fort.

Für eine gegebene Distanzfunktion *dist* ist die Funktion *median* folgendermaßen definiert:

$$\text{median}(\{m_1, \dots, m_k\}) := \arg \min_{m \in \{m_1, \dots, m_k\}} \left(\sum_{i=1}^k \text{dist}(m, m_i) \right)$$

Die Distanzfunktion wird dem Konstruktor der Klasse *KMedian* in Form eines Objektes übergeben, dessen Klasse das Interface *Metric* implementiert. Dieses Interface stellt die Methode *distance(P e1, P e2)* zur Verfügung.

In dieser Aufgabe wählen wir als Abbruchkriterium schlicht, dass der Algorithmus *n* Iterationen durchlaufen hat. Wie die Menge *E* und der Wert *k* ist auch *n* ein Parameter der Methode *cluster(Collection e, int k, int n)*.

Vervollständigen Sie die Klasse an allen Orten, die mit einem //TODO markiert sind.

```
public class KMedian<P> {
    private final Metric<P> metric;
    public KMedian(Metric<P> metric) {
        this.metric = metric;
    }
    public Map<P, Set<P>> cluster(Collection<P> e, int k, int n) {
        // Create copy of set first
```

```

    ArrayList<P> elements = new ArrayList<P>();
    elements.addAll(e);
    Random rnd = new Random();
    if (elements.size() < k) {
        // k is too large for the given set e
        return null;
    }
    ArrayList<P> centers;
    // TODO Choose centers randomly
    Map<P, Set<P>> clustering = null;
    for (int i = 0; i < n; i++) {
        // TODO Cluster
    }
    return clustering;
}
/* cluster the elements s.t. each element is mapped to the nearest center */
public Map<P, Set<P>> cluster(Collection<P> centers, Collection<P> elements) {
    // TODO
    return null;
}
/* find the new centers, i.e., calculate the median for each cluster */
public ArrayList<P> findNewCenters(Map<P, Set<P>> clustering) {
    // TODO
    return null;
}
}

```

Aufgabe H14 (5 Punkte)²

Implementieren Sie eine Klasse *MapInverter*, die eine Methode zur Inversion einer *Map* zur Verfügung stellt. Eine Inversion ist hierbei eine neue *Map*, bei der die *values* auf die *keys* der alten *Map* abgebildet werden. Da die ursprüngliche *Map* nicht zwangsläufig injektiv ist¹, muss das Ergebnis der Methode eine *Map* des Typs *Map<B, Set<A>>* sein, wobei die übergebene *Map* den Typ *Map<A, B>* hat.

Die Methode soll also folgende Signatur haben:

```
public java.util.Map<B, java.util.Set<A>> invert(java.util.Map<A, B> map)
```

Sie können bei der Lösung der Aufgabe davon ausgehen, dass alle Objekte des Typs *A* oder *B* sinnvolle Implementierungen der Methoden *hashCode* und *equals* zur Verfügung stellen.

Aufgabe H15 (5 Punkte)

In dieser Aufgabe sollen Sie beim Entwurf eines EDV-Systems für einen Fahrzeughändler mithelfen. Der Fahrzeughändler handelt mit

- Fahrrädern,

¹Das heißt es kann sein, dass die *Map* unterschiedlichen *keys* den gleichen *value* zuordnet, wobei zwei Valueobjekte *o1* und *o2* in diesem Kontext als "gleich" betrachtet werden, wenn *o1.equals(o2)* gilt.

- Einrädern,
- Mopeds,
- Motorrollern,
- PKWs,
- LKWs,
- Kutschen,
- Draisinen und
- Zügen.

Ihre Kollegen haben bereits festgelegt, dass zur Umsetzung des Systems folgende Typen benötigt werden:

- ein Typ für jede gehandelte Fahrzeugkategorie,
- *Fahrzeug*,
- *Helmpflichtig* (Beachten Sie, dass die Helmpflicht für Ein- und Fahrräder zwischenzeitlich eingeführt wurde!),
- *Zweiraedrig*,
- *MotorisiertesFahrzeug*,
- *Vierraedrig*,
- *UnmotorisiertesFahrzeug*,
- *PersonenTransportfaehig* (für Fahrzeuge, die dafür ausgelegt sind, neben dem Fahrer mindestens zwei weitere Personen zu transportieren) und
- *Pedalgetrieben*

Welche dieser Typen würden Sie als Interfaces, welche als abstrakte Klassen und welche als konkrete Klassen umsetzen? Geben Sie jeweils eine kurze Begründung für Ihre Entscheidung an! Falls Ihre Begründungen in mehreren Fällen gleich sind, ist es ausreichend, die Begründung einmal anzugeben, falls deutlich wird, für welche Typen sie zutrifft.

Aufgabe H16 (6 Punkte)

Betrachten Sie die folgende Klasse *Element* $\langle E \rangle$, welche ein Element einer zyklischen Liste implementiert und dabei eine innere Klasse *Parity* bereitstellt, welche gerade und ungerade Zahlen voneinander unterscheiden, sonst aber alle Zahlen als gleich ansehen soll.

```
import java.util.*;
public class Element<E> {
    public static class Parity {
        private int value;
```

```

    public Parity(int v) {
        this.value = v;
    }
    public boolean equals(Object o) {
        return o instanceof Parity && ((Parity) o).value%2 == this.value%2;
    }
    public int getCreationValue() {
        return this.value;
    }
    public int hashCode() {
        return this.value%3;
    }
    public boolean isEven() {
        return this.value%2 == 0;
    }
    public String toString() {
        return " " + this.value;
    }
}
public static void main(String[] args) {
    Element<Parity> e = new Element<Parity>(new Parity(1));
    e.add(new Parity(2));
    e.add(new Parity(3));
    System.out.println(e.contains(new Parity(2)));
    Set<Parity> set = new LinkedHashSet<Parity>();
    set.add(new Parity(1));
    set.add(new Parity(2));
    Parity p = new Parity(3);
    if (!set.contains(p)) {
        for (Parity o : set) {
            if (o.equals(p)) {
                System.out.println("This must not happen!");
            }
        }
    }
    set.add(p);
    set.add(new Parity(0));
    set.add(new Parity(6));
    System.out.println(set);
}
private Element<E> next;
private final E value;
public Element(E v) {
    assert (v != null) : "Value must not be null!";
    this.value = v;
    this.next = this;
}
public void add(E v) {

```

```

    Element<E> e = new Element<E>(v);
    e.next = this.next;
    this.next = e;
}
public boolean contains(Object o) {
    return this.value.equals(o) || this.next.contains(this, o);
}
public boolean equals(Object o) {
    return o instanceof Element && this.value.equals(((Element<?>) o).value);
}
public Element<E> getNext() {
    return this.next;
}
public E getValue() {
    return this.value;
}
public int hashCode() {
    return 3 * this.value.hashCode();
}
private boolean contains(Element<E> origin, Object o) {
    return !this.equals(origin) && (this.value.equals(o) || this.next.contains(origin, o));
}
}

```

Wie lautet die Ausgabe dieses Programms bei Ausführung der *main* Methode und warum? Welche Fehler sind in diesem Programm enthalten und wie kann man sie mit Änderungen an genau zwei Zeilen beheben (die *main* Methode soll dabei unverändert bleiben)? Wie lautet die Ausgabe der *main* Methode im korrigierten Programm?

Aufgabe H17 (10 Punkte)²

Auf der Webseite der Vorlesung ist eine Implementierung von k -Medians für Bilder hinterlegt sowie eine Sammlung von Verkehrsschildern.

Implementieren Sie eine Distanzfunktion für Bilder—insbesondere die mitgelieferten Verkehrsschilder—welche ihre Ähnlichkeit möglichst gut widerspiegelt. Implementieren Sie dazu eine Klasse *ImageMetric* welche das Interface *lufti.kmedians.Metric<BufferedImage>* implementiert. Die Klasse *java.awt.image.BufferedImage* wird von Java bereitgestellt, um mit Bitmaps zu arbeiten. Wir empfehlen, die Methoden *getWidth*, *getHeight* und *getRGB* dieser Klassen nachzuschlagen. Zur Bearbeitung von Farbwerten bietet sich die Klasse *java.awt.Color* an.

So können Sie z.B. den Blauanteil des Pixels mit den Koordinaten (17, 34) feststellen:

```

import java.awt.Color;
import java.awt.image.BufferedImage;
...
int rgb = image.getRGB(17, 34);
Color col = new Color(rgb);
int blauAnteil = col.getBlue();

```

Erläutern Sie kurz, warum Sie diese Distanzfunktion gewählt haben.

Verwenden Sie den Klassifikator aus der Tutorübung, um die vorgegebenen Verkehrszeichen in 2, 3, 4 oder 5 Mengen zu klassifizieren.

Was erhalten Sie als Ergebnis? Sind Sie damit zufrieden?

Abgabe zum 17.12.2013

²Bitte Quelltext für die Abgabe ausdrucken und zusätzlich per E-mail an den jeweiligen Tutor senden.