technische universität
dortmund

Fakultät für Informatik
Lehrstühle 2, 11

Prof. Dr. C. Sohler, A. Krivošija
H. Blom, N. Kriege, H. Timm, B. Zey
http://tiny.cc/dap2p

Sommersemester 2014

DAP2 Praktikum – Blatt 4

Ausgabe: 28. April — Abgabe: 05.–09. Mai

Studienleistung (Scheinkriterien)

- Zum bestehen des Praktikums muss jeder Teilnehmer die folgenden Leistungen erbringen
  - Es müssen mindestens 50 Prozent der Punkte in den Kurzaufgaben erreicht werden.
  - Es müssen mindestens 50 Prozent der Punkte in den Langaufgaben erreicht werden.
  - Man muss an mindestens 12 Terminen Anwesend sein, und aktiv mitarbeiten (Nach Absprache mit den Tutoren ist es auch möglich, Fehltermine in anderen Gruppen nachzuholen, sofern dort Platz ist).
- An Feiertagen suchen Sie sich bitte einen Ersatztermin, ansonsten wird dies als "nicht erschienen"gewertet.

Hinweis zu den Langaufgaben

Für diese Aufgaben stehen Ihnen zwei Wochen Bearbeitungszeit zur Verfügung. In den Praktika dieser Woche können (und sollen) Sie mit der Bearbeitung beginnen. Den Rest müssen Sie außerhalb des Praktikums erledigen. Neben den Lernräumen können Sie dafür auch die Poolräume (z. B. in einer Freistunde) nutzen, wenn dort gerade keine Veranstaltungen stattfinden. Im Praktikum in zwei Wochen präsentieren Sie Ihrem Tutor dann ihre Lösungen auf Ihrem Praktikumsrechner im Pool und bekommen Punkte dafür (nach Absprache mit dem Tutor ist es auch möglich, die Lösung auf ihrem eigenen Rechner zu präsentieren). **Wichtig:** Sie müssen die Aufgaben **vor** Beginn des Praktikums (in dem Sie die Lösungen präsentieren) fertig haben. `Sie dürfen die Langaufgaben in Gruppen von bis zu drei Studierenden bearbeiten und präsentieren.`

Ebenfalls wichtig: Der Quellcode ist natürlich mit sinnvollen Kommentaren und die Schleifen mit zugehörigen Invarianten (inkl. Bereich in dem die Invariante gilt) zu kommentieren.

Langaufgabe 4.1 **(4 Punkte)**

Lernziel: Euklidischer Abstand

Diese Aufgabe besteht aus mehreren Teilen. Bitte lesen Sie sich die Aufgabe vorher komplett durch und überprüfen Sie nach der Bearbeitung, ob Sie nichts vergessen haben.

- Legen Sie eine Klasse `Point` an, die einen Punkt im $\mathbb{R}^d$ beschreiben soll. Auch negative Werte und 0 sind für die jeweiligen Koordinaten zulässig.
- Die Klasse `Point` soll einen Konstruktor bereitstellen, der die Dimension d (`int`) und die Werte der einzelnen Dimensionen (`double... values`) entsprechend definiert. Dieser Konstruktor soll eine IllegalArgumentException werfen, wenn fehlerhafte Parameter übergeben werden.
  Hinweis dazu: Suchen sie nach „Java Varargs“ in der Suchmaschine ihrer Wahl.
- Schreiben sie eine Methode `public double get(int i)` und die Methode `public int dim()`, die die Werte der jeweilige Koordinate zurück gibt und die Dimensionalität des Punktes angibt.

  Für eine ausgiebige Implementierung eines Punktes bzw. Vektors siehe:
  `http://commons.apache.org/proper/commons-math/apidocs/org/apache/commons/math3/linear/RealVector.html`
- Legen Sie außerdem eine abstrakte Klasse `Simplex` an, die $d+1$ dieser Punkte enthält. Die Klasse soll einen Konstruktor bereitstellen, der die Dimension d (`int`) und die $d+1$ Punkte (`Point... points`) entsprechend definiert. Diese geben ihre Eckpunkte an. Sie sollten auch eine passende Get-Methode schreiben, um die jeweiligen Punkte zurück zu geben. Legen Sie eine abstrakte Methode `public abstract boolean validate()` an welche überprüfen soll, ob die jeweilige Instanz einen validen Simplex darstellt.
- Implementieren Sie eine Klasse `Triangle`, die von Simplex erbt. Die `validate`-Methode soll überprüfen, dass 3 Punkte vorhanden sind und diese jeweils genau Dimension 2 haben (d.h. $d = 2$).
- Schreiben Sie in die Klasse `Simplex` eine weitere Methode `public double perimeter()`, die die Summe der Seitenlängen des Simplex zurück gibt.
- Definieren Sie ein Interface `Distance` mit einer Methode `public double distance(Point p1, Point p2)`, welche den Abstand zweier Punkte berechnet. Implementieren Sie dieses Interface in der Klasse `EuclidDistance`, so dass die euklidische Distanz von der Methode `public double distance(Point p1, Point p2)` berechnet wird.
- Schreiben Sie eine Klasse `Application`, und fügen Sie dieser eine `main`-Methode hinzu.
- Die `main` Methode soll entweder 6 Parameter annehmen, daraus ein Dreieck bauen und dessen Umfang dann ausgeben, oder ohne Parameter ein zufälliges Dreieck erstellen und dessen Umfang ausgeben. Die Koordinaten der Punkte dieser Dreiecke sollten sich aus Gründen der Nachvollziehbarkeit im Interval [-1000; 1000] befinden. Falls Sie unsicher sind wie das funktioniert, lesen Sie noch einmal die Hinweise zu Zufallszahlen.
- Beispielaufruf: `java Application x1 y1 x2 y2 x3 y3`

Langaufgabe 4.2 **(4 Punkte)**
Lernziel: Convex Hull

- Schreiben Sie eine Klasse `ConvexHull`, die für eine gegebene Menge an Punkten `P` im $\mathbb{R}^2$, die konvexe Hülle berechnet.
- Implementieren Sie die einfache iterative Methode, die in der Vorlesung als `SimpleConvexHull(P)` eingeführt wurde. Nennen Sie die Methode `public List<Point> simpleConvex(Point[] P)`.
- Es müssen für je zwei Punkte aus `P` eine Gerade berechnet und entschieden werden ob alle restlichen Punkte links oder rechts der Geraden liegen.
- Zum Schluss erzeugen Sie, wie in der Vorlesung beschreiben, eine verkettete Liste, die alle Punkte der konvexen Hülle enthält.
- Berechnen Sie die konvexe Hülle für 1000 zufällig erzeugte Punkte.
- Hinweis: Zum Testen können Sie z. B. ein Dreieck mit den Eckpunkten [(10,10), (10,100), (100,10)] verwenden und dann nur Punkte generieren, die innerhalb dieses Dreiecks liegen.
- Geben Sie die Punkte der konvexen Hülle in der entsprechenden Reihenfolge auf den Bildschirm aus. Wie kann man das Resultat überprüfen?

Beachten Sie die Hinweise und Tipps auf den folgenden Seiten.

Hinweise und Tipps

4.1 Zufallszahlen (noch einmal)

In Java steht ein Pseudozufallszahlengenerator zur Verfügung. Die Klasse `java.util.Random` stellt den Konstuktor für einen Pseudozufallszahlengenerator, sowie die Methode `nextInt()` zur Verfügung.

Um Zufallszahlen zu erzeugen, kann wie folgt vorgegangen werden:

```
...
// Den Generator erzeugen (als Seedwert wird die Systemzeit verwendet)
java.util.Random numberGenerator = new java.util.Random();
...
//Wann immer man eine Zufallszahl zwischen 0 und 1 braucht
double randomNumber = numberGenerator.nextDouble();
...
```

Um Zufallszahlen in bestimmte Bereiche einzugrenzen, geht man wie folgt vor:

```
int grenze = 1000;
...
//Wann immer man eine Zufallszahl braucht
double randomNumber = numberGenerator.nextDouble() * grenze;
if(numberGenerator.nextBoolean())
  randomNumber *= -1;
```

Damit würden die Zufallszahlen immer im Bereich [-grenze, grenze] liegen.

Siehe auch:

`http://docs.oracle.com/javase/7/docs/api/java/util/Random.html`

Für eine andere Implementierung eines Pseudozufallszahlengenerators siehe: `http://commons.apache.org/proper/commons-math/apidocs/org/apache/commons/math3/random/MersenneTwister.html`

4.2 Convex Hull Algorithmus

Sie finden unter folgendem Link eine ausführliche Beschreibung des Simple Convex Hull Algorithmus:

`http://ls2-www.cs.uni-dortmund.de/lehre/sommer2014/dap2/folien/Vorlesung07.pdf`

Falls die Folien noch nicht rechtzeitig online sind können Sie die Beschreibung in den Folien aus dem letzten Jahr finden:

`http://ls2-www.cs.uni-dortmund.de/lehre/sommer2013/dap2/folien/Vorlesung07.pdf`

4.3 Verwendung von IDEs

Für das Praktikum wird es ausreichend sein, die Aufgaben mit einem Texteditor und dem Kommandozeilen-Compiler/Interpreter zu lasen. Es ist Ihnen zwar erlaubt eine beliebige IDE (Eclipse, NetBeans, etc.) zu verwenden, allerdings sind Sie dann selbst dafür verantwortlich diese einzurichten, und sicherzustellen, dass diese auch funktioniert. Es ist nicht Aufgabe der Tutoren Ihnen bei Problemen, die durch die Verwendung einer IDE entstehen, zu helfen.