technische universität dortmund

Fakultät für Informatik
Lehrstühle 2, 8, 12

Prof. apl Dr. Beate Bollig
K. Brandenbusch, P. Oberdiek, E. Thordsen

Sommersemester 2020
8. Mai

DAP2 Praktikum – Blatt 4

Abgabe: 18. Mai, 14:00 Uhr

Langaufgabe 4.1: Konvexe Hüllen mit Teile & Herrsche (8 Punkte)

Diese Aufgabe besteht aus mehreren Teilen. Bitte lesen Sie sich die Aufgabe vorher **komplett** durch und überprüfen Sie alle Teile nach der Bearbeitung, damit Sie nichts vergessen!

Im Rahmen dieser Aufgabe benötigen Sie die folgenden Klassen:

- `Point`:
  - Die Klasse `Point` soll es ermöglichen, Punkte im $\mathbb{R}^d$ zu repräsentieren. Zu diesem Zweck müssen die Koordinaten der Punkte darin als Gleitkommazahlen gespeichert werden können. Auch negative Werte und 0 sind für die jeweiligen Koordinaten zulässig.
  - Der Konstruktor von `Point` soll eine beliebige Anzahl an Argumenten des Typs `double` akzeptieren.
    Hinweis: Suchen Sie nach „Java Varargs“ in der Suchmaschine Ihrer Wahl.
  - Die Klasse `Point` soll außerdem die Methoden `equals` und `toString` der allgemeinen Superklasse `Object` überschreiben. Dabei soll `equals true` zurück geben, gdw. alle Koordinaten mit denen des übergebenen Objektes übereinstimmen. Die `toString` Methode soll einen String ausgeben, der die Koordinaten des Punktes durch Leerzeichen getrennt enthält (z.B. „`13.123 5.4123 4.1`“). Die Ausgabe soll auf mindestens 4 Nachkommastellen präzise sein. Ob Sie Nullen am Ende der Zahlen darstellen oder nicht, ist Ihnen überlassen.
  - Über die Methode `get(int i)` soll die `i`-te Koordinate abgefragt werden können.
- `PointsGenerator`:
  - Die Klasse `PointsGenerator` soll im Konstruktor zwei Argumente entgegen nehmen: Eine untere und eine obere Grenze für Koordinaten `min` und `max`.
  - Es soll eine Methode `generate` bereitstehen, die eine positive ganze Zahl entgegen nimmt und eine entsprechende Menge an zufälligen **2-dimensionalen** Punkten aus $[\texttt{min}, \texttt{max}]^2$ als Liste zurück gibt.
    Hinweis: Sehen Sie sich die Klasse `java.util.ArrayList` an. Diese ist eine brauchbare Implementierung von Listen.

- `Line`
  - Die Klasse `Line` soll einen Konstruktor bereitstellen, der zwei `Point`-Objekte akzeptiert. Der erste Punkt ist der Startpunkt und der zweite der Endpunkt einer Linie.
  - In dieser Klasse soll eine Methode `side` bereitstehen, die ein `Point`-Objekt akzeptiert und die folgenden Ausgaben macht:

    0 Wenn der übergebene Punkt auf der Geraden durch den Start- und Endpunkt liegt

    1 Wenn der übergebene Punkt „links“ der Geraden liegt.

    -1 Wenn der übergebene Punkt „rechts“ der Geraden liegt.

    Die Blickrichtung ist dabei vom Start- zum Endpunkt. Ist der Startpunkt $(0, 0)$ und der Endpunkt $(1, 0)$, so liegt der Punkt $(-1, 1)$ „links“ von der Geraden und es müsste 1 zurückgegeben werden. Werden Start- und Endpunkt vertauscht, so invertieren sich auch die Ausgaben von `side`.
    Hinweis: Geben Sie beispielsweise „side of line“ in die Suchmaschine Ihrer Wahl ein.

- `ConvexHull`
  - Die Klasse `ConvexHull` soll eine Methode `computeHull` bereitstellen, die eine Liste von **2-dimensionalen** `Point`-Objekten akzeptiert und eine Liste von **2-dimensionalen** `Point`-Objekten zurück gibt.
  - Die zurückgegebenen Punkte sollen die Eckpunkte der konvexen Hülle der übergebenen Punkte sein.
  - Um die konvexe Hülle zu berechnen, sollen Sie den „Teile & Herrsche“-Ansatz verwenden, dessen Pseudocode Sie weiter unten finden.
  - Andere Algorithmen zur Berechnung der konvexen Hülle werden **nicht** akzeptiert.

- `ConvexTester`
  - Mittels dieser Klasse soll Ihr Code ausführbar gemacht werden. Daher soll die Klasse `ConvexTester` eine `main`-Methode enthalten.
  - In der `main`-Methode sollen Sie **exakt** drei Zahlen als Argumente akzeptieren:
    (a) Die untere Schranke für Koordinaten
    (b) Die obere Schranke für Koordinaten
    (c) Die Anzahl zu generierender Punkte
  - Stellen Sie sicher, dass die Parameter sinnvolle Werte enthalten.
  - Die `main`-Methode soll zunächst mittels `PointsGenerator` so viele Punkte generieren, wie per Argumente übergeben wurden und anschließend die konvexe Hülle mittels `ConvexHull` berechnen.
  - Anschließend sollen Sie zählen, wie viele Punkte nicht in der konvexen Hülle liegen.
  - Sie sollen anschließend die folgenden Ausgaben in dieser Reihenfolge in die Kommandozeile schreiben:
    (a) Die Anzahl generierter Punkte
    (b) Die Koordinaten der Punkte mit einem Punkt pro Zeile (s. `toString` oben)
    (c) Die Anzahl Punkte in der konvexen Hülle

(d) Die Koordinaten der Punkte der konvexen Hülle mit einem Punkt pro Zeile (s. `toString` oben)

(e) Die Zeile „Alle Punkte sind in der Huelle“, wenn dies zutrifft, und die Zeile „x Punkte liegen nicht in der Huelle“, wenn x Punkte nicht in der berechneten konvexen Hülle liegen.[1]

Dabei sollten Sie die folgenden allgemeinen Bedingungen beachten:

- Fehlermeldungen jedweder Art sollen wie zuvor mit „FEHLER:“ beginnen.
- Das Ausgabeformat muss **exakt** eingehalten werden.
- Die Korrektheit Ihres Codes soll an geeigneten Stellen über Assertions sichergestellt werden. Auch diese gehen in die Bewertung ein! Assertions sollen **nicht** zur Überprüfung von Nutzereingaben verwendet werden.
- Beachten Sie die Hinweise am Ende des Blattes.

Der zu implementierende Algorithmus zur Berechnung konvexer Hüllen nutzt einen rekursiven Aufruf, der ähnlich zu `MergeSort` ist. Dabei werden die Punkte wiederholt unterteilt, bis sie klein genug sind, um eine konvexe Hülle unmittelbar durch *brute force* zu ermitteln. Der Algorithmus nutzt dabei aus, dass die untersuchten Punkte nur zwei Dimensionen haben und berechnet zuerst die „obere Hälfte“ der konvexen Hülle und anschließend die „untere Hälfte“ der konvexen Hülle. Die beiden Hälften der konvexen Hülle müssen anschließend zusammengesetzt werden. Der Algorithmus für die obere Hälfte sieht dabei wie folgt aus:

- Sortiere alle Punkte nach aufsteigender erster Koordinate
- Unterteile die Punkte in linke und rechte Hälften, bis die Intervalle nur noch aus maximal 3 Punkten bestehen
- Für bis zu 3 Punkte kann die obere konvexe Hülle direkt berechnet werden. Doppelt vorkommende Punkte können auf ein Vorkommen reduziert werden. Bei bis zu 2 verschiedenen Punkten ergeben diese die obere konvexe Hülle. Bei 3 verschiedenen Punkten, muss der mittlere verworfen werden, falls dieser „unterhalb“ der Verbindung des ersten und dritten Punktes liegt.
- Um zwei benachbarte konvexe Hüllen zu kombinieren, untersuche die Verbindung der beiden benachbarten Endpunkte. Solange einer der Vorgänger/Nachfolger der Endpunkte auf der konvexen Hülle „oberhalb“ dieser Verbindungslinie liegt, verwerfe den entsprechenden Endpunkt und bilde eine neue Verbindung zu dessen Vorgänger/Nachfolger. Dabei soll das Ergebnis eine „nach rechts gekrümmte“ Punktfolge ergeben.
- Verbinde nach diesem Verfahren alle zuvor vorgenommenen Unterteilungen, bis eine obere konvexe Hülle für alle Punkte berechnet ist.

Die Formulierung in diesem Pseudocode ist nicht rekursiv, damit Sie in einem Pseudocode zusammengefasst werden kann. Sie sollten die Implementierung jedoch rekursiv vornehmen.[2]

[1] Der Wert x ist hier ein Platzhalter! Bei zwei Punkten soll die Ausgabe „2 Punkte ...“, bei drei Punkte „3 Punkte ...“ usw. lauten.

[2] Eine alternative Formulierung mit entsprechenden Grafiken finden Sie auch im Foliensatz 05 der Vorlesung.

Der Algorithmus für die untere Hälfte ist analog, wobei die Bedeutung von „oberhalb“ und „unterhalb“ vertauscht werden muss. Beachten Sie, dass die Bedeutung von „links“ und „rechts“ in der Klasse `Line` von der Reihenfolge der Punkte abhängt. Das können Sie ausnutzen, um die untere Hälfte zu berechnen.

Beachten Sie zudem, dass beim Zusammenfügen der oberen und unteren Hälfte möglicherweise doppelte Punkte entfernt werden müssen! Die Ausgabe der Punkte der konvexen Hülle soll keine Duplikate und ausschließlich Eckpunkte der konvexen Hülle enthalten!

Beispiele:

```
Input:  java ConvexTester -3.14 3.14 7
Output: 7
        2.2120692671941433 2.4339863707624665
        2.1815189930584755 -1.518639819487326
        2.056360341317227 -1.0405335654808945
        1.960310553932651 -0.637061491951842
        1.6813392882423925 2.2059187687021944
        1.6782664366680184 -1.424269677768478
        -0.22125397653447676 -1.4519886459977598
        4
        -0.22125397653447676 -1.4519886459977598
        1.6813392882423925 2.2059187687021944
        2.2120692671941433 2.4339863707624665
        2.1815189930584755 -1.518639819487326
        Alle Punkte sind in der Huelle

Input:  java ConvexTester 1 2 3
Output: 3
        1.3621839863768272 1.9960858023195986
        1.3561448251803259 1.7148874563542933
        1.3148380014103105 1.9469087049358116
        3
        1.3148380014103105 1.9469087049358116
        1.3621839863768272 1.9960858023195986
        1.3561448251803259 1.7148874563542933
        Alle Punkte sind in der Huelle
```

Hinweise und Tipps

Zufallszahlen (noch einmal)

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

Um Zufallszahlen zu erzeugen, kann wie folgt vorgegangen werden:

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

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

```
double lowerBound = 3.14;
double upperBound = 9.65;
...
// Wann immer man eine Zufallszahl braucht:
double randomNumber = lowerBound
                    + (upperBound - lowerBound) * generator.nextDouble();
```

Damit werden Zufallszahlen zwischen `untereGrenze` (inklusive) und `obereGrenze` (exklusive) generiert. Für ganze Zahlen kann alternativ die Methode `nextInt(grenze)` verwendet werden, die eine ganze Zufallszahl zwischen 0 (inklusive) und `grenze` (exklusive) generiert.

Siehe auch: `http://docs.oracle.com/javase/8/docs/api/java/util/Random.html`