FAU, Informatik 2, AUD-Team
aud@i2.cs.fau.de

4. Übung

Abgabe bis 29.05.2017, 10:00 Uhr

Einzelaufgabe 4.1: *KnotPoint 2017S* **12 EP**

Herzlichen Glückwunsch zum Erwerb Ihres neuen *KnotPoint 2017S* – dem programmierbaren autonomen Staubsauger aus dem Hause *AuD*! Für einen sicheren Betrieb Ihres $\mathcal{KP}$ beachten Sie bitte unbedingt unsere autonomen Grundreinigungsbestimmungen (kurz: „AGB“):

§ 1 Ihr $\mathcal{KP}$ wird ausschließlich in `PHP` ($\mathcal{P}$rogrammable $\mathcal{H}$oovering $\mathcal{P}$rocessor) programmiert. Als treuer $\mathcal{AuD}$-Kunde steht Ihnen in unserem Download-Bereich eine Vorlage zur Verfügung.

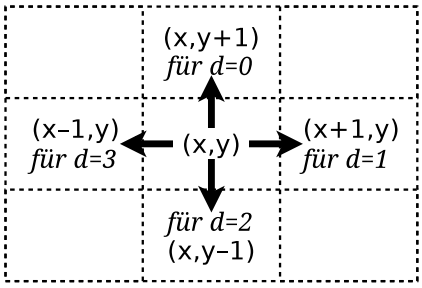
§ 2 Ihr $\mathcal{KP}$ verfügt über einen eingebauten Akku mit Ladestandsanzeige und einen Staubbehälter mit Füllsensor. Zum Lieferumfang gehört eine fest zu montierende Ladestation: Ihr $\mathcal{KP}$ lässt sich nur in dieser Station einschalten und muss selbständig zur Station zurückkehren. ***Achtung:*** Stellen Sie daher sicher, dass Ihr `PHP`-Code ***keinen Zustand*** von einem `hoover()` zum nächsten behält – reinitialisieren Sie ggf. alle Attribute am Anfang von `hoover()`!

§ 3 Ihr $\mathcal{KP}$ hat keinerlei Orientierungshilfe, daher muss Ihr `PHP`-Code die bereits abgesaugten Felder sowie die Position und Ausrichtung des Gerätes selbst verwalten. Zu diesem Zweck teilt $\mathcal{KP}$ den Raum virtuell in ein Gitter aus gleichmäßigen Feldern ein (siehe § 5).

§ 4 Ihr `PHP`-Code kann mit der realen $\mathcal{KP}$-Hardware *ausschließlich* über die vorgegebene Schnittstelle interagieren. Diese besteht aus folgenden Methoden:

- ▶ `move()`: fährt $\mathcal{KP}$ ein Feld vorwärts
- ▶ `turnClockwise()`: dreht $\mathcal{KP}$ um $90°$ im Uhrzeigersinn
- ▶ `moveBack()`: fährt $\mathcal{KP}$ ein Feld rückwärts (der abgefahrene Weg bleibt garantiert frei)
- ◁ `hasObstacle()`: prüft, ob sich im Feld unmittelbar vor $\mathcal{KP}$ ein Hindernis befindet
- ◁ `battery()`: gibt den Ladezustand des Akkus zurück (entspricht der Anzahl der ab jetzt noch möglichen `move`- bzw. `moveBack`-Aufrufe – `turnClockwise()` wirkt sich nicht auf den Ladezustand aus, dennoch darf die Batterie beim Aufruf nicht leer sein!)
- ◁ `binFull()`: prüft, ob der Staubbehälter voll ist

§ 5 Nur wenn im Feld vor $\mathcal{KP}$ kein Hindernis liegt, darf $\mathcal{KP}$ das Feld betreten und bearbeiten.
Nebenstehende Grafik zeigt exemplarisch das Koordinatensystem des $\mathcal{KP}$: Befindet sich das Gerät gerade im Feld (x, y) mit der Ausrichtung $d = 1$, dann ergibt sich die neue Position nach einem Aufruf von `move` zu $(x + 1, y)$ und die neue Ausrichtung nach einem Aufruf von `turnClockwise()` zu $d == 2$.



§ 6 Für ein optimales Ergebnis ***darf*** Ihr $\mathcal{KP}$ jedes Feld genau einmal saugen (d.h. mit `move` betreten) und bereits bearbeitete Felder nur noch auf seinem Rückweg (`moveBack`) überfahren.

§ 7 Sobald die Akku-Restladung gerade noch für den Rückweg ausreicht, der Staubbehälter voll ist oder alle erreichbaren Felder gesaugt sind, ***muss*** Ihr $\mathcal{KP}$ ***sofort*** den Rückweg antreten und *exakt* in die Ausgangsstellung zur Ladestation zurückkehren.

§ 8 Wegen der sehr beschränkten Kapazitäten und Ressourcen Ihres $\mathcal{KP}$ darf Ihr `PHP`-Code aus der Java-API *ausschließlich* die Klasse `String` verwenden.

Steuern Sie $\mathcal{KP}$ so, dass er nicht Amok läuft, alle AGB beachtet und dabei gründlich staubsaugt. Implementieren Sie dazu die Hilfsfunktionen in `PHP` ($\mathcal{P}$rogrammable $\mathcal{H}$oovering $\mathcal{P}$rocessor) gemäß vorhandenem Kommentar und idealerweise in der vorgegebenen Reihenfolge.
Laden Sie Ihre `PHP.java` zum Testen zuerst in den *KnotPoint 2017S* und dann ins EST.

Einzelaufgabe 4.2: Dynamische Programmierung — 13 EP

Die *Rowland*-Folge hat eine interessante Eigenschaft: Berechnet man die Differenz paarweise aufeinander folgender Werte und überspringt davon alle „1"-Glieder, so bleibt eine *Primzahlen-Folge* übrig. Leider ist die naiv-rekursive Implementierung bereits ab dem 7. Glied untragbar langsam.
Mit dem Konzept der „*Dynamischen Programmierung*" mittels *Memoization* (kurz: DP) haben Sie ein Verfahren kennengelernt, um doppelte Berechnungen gänzlich zu vermeiden. Legen Sie eine Klasse `Rowland` mit den folgenden öffentlichen Klassenmethoden an:

a) `int rowlandNaive(GCD gcd, int n)` berechnet den n-ten Wert der *Rowland*-Folge naiv-rekursiv ohne weitere Optimierungen nach der Vorschrift von *OEIS*:
► *a(1) = 7; for n > 1, a(n) = a(n-1) + gcd(n, a(n-1))*

b) `int omitNaive(GCD gcd, int n)` verwendet die vorangehende `rowlandNaive` um die ersten n *Rowland-Primzahlen* ebenfalls naiv und ohne weitere Optimierungen zu berechnen.

c) Implementieren Sie `int rowlandDP(GCD gcd, int n)` analog zu `rowlandNaive` *ebenfalls baumrekursiv*, aber brechen Sie hier nun die Rekursion ab, sobald Sie auf ein Glied stoßen, dass Sie bereits berechnet haben.
Verwalten Sie die Folge in einem privaten `int[]`-Feld innerhalb Ihrer Klasse. Beachten Sie bitte, dass die erforderliche Größe des Feldes nicht vorab bekannt ist – stattdessen müssen Sie das Feld *nach Bedarf optimal anlegen bzw. nachträglich vergrößern*, **ohne** die alten Werte zu verlieren (siehe `System.arraycopy`). Stellen Sie zusätzlich eine Methode `static void resetDP()` bereit, die das Feld auf `null` zurücksetzt und dadurch alle bisher errechneten *Rowland*-Glieder vernichtet.

d) Vervollständigen Sie schließlich die Klasse `Rowland` um eine Methode `int omitDP(GCD gcd, int n)`, die wie `omitNaive` funktioniert, aber ebenfalls von der *Memoization* in `rowlandDP` profitiert.

Achtung - Wichtig: Verwenden Sie für die Berechnung des ggTs **ausschließlich** die bereitgestellte Methode `GCD.gcd(int,int)`! Sie dürfen **keine** zusätzlichen Methoden in Ihrer Klasse `Rowland` deklarieren! Ihre Klasse muss **genau ein** privates `int[]`-Feld enthalten! Sie dürfen **keine** Methoden oder Klassen aus der Java-API verwenden! Beachten Sie **unbedingt den öffentlichen** Testfall!
Geben Sie Ihre Datei `Rowland.java` über EST ab.

Gruppenaufgabe 4.3: Closest Pair — 19 GP

In dieser Aufgabe sollen Sie eine Klasse `ClosestPair` mit drei Methoden implementieren, die aus einer Liste von Punkten in der Ebene das Paar mit der geringsten euklidischen Distanz ermitteln. Die vorgegebene Schnittstelle `ClosestPairHelper` enthält einige Hilfsmethoden, Konstanten, Vereinbarungen und Kommentare, die Ihnen die Implementierung deutlich erleichtern – eine Instanz davon wird jeder Ihrer Methoden über das erste Argument `cph` aus den Testfällen übergeben.

a) Erstellen Sie eine ***rekursive*** Hilfsmethode `closestPointHelper(cph, p, pd, ps)`:
Sie erhält den festen Startpunkt `p`, für den der Punkt mit minimalem euklidischem Abstand gefunden werden soll. `pd` ist dabei derjenige Punkt (kombiniert mit seiner Distanz zu `p` ⇝ siehe `ClosestPairHelper`), der bisher den kleinsten bekannten Abstand zu `p` hat. Die Punkteliste `ps` enthält alle noch zu untersuchenden Punkte. Falls `ps` leer ist, liefert die Methode als

Ergebnis den Punkt `pd` mit dessen Distanz zurück. In allen anderen Fällen liefert diese Methode denjenigen Punkt (ebenfalls jeweils mit Abstand zu `p`) aus `ps` und `pd` zurück, der die minimale euklidische Distanz zu `p` hat.

b) Implementieren Sie nun die (nicht-rekursiven) Methode `closestPoint(cph, p, ps)`:
Anstelle von `closestPointHelper` startet ein Benutzer die Suche nach einem Punkt mit minimaler Distanz zu `p` durch `closestPoint`, die intern hier **unbedingt** die vorangehend erstellte `closestPointHelper` **verwenden muss**. Für die Umsetzung eines rekursiven Verfahrens benötigt man oft eine Hilfsmethode (hier `closestPointHelper`), da für die Rekursion oft mehr Methodenparameter benötigt werden, als die Hauptmethode (hier `closestPoint`) erlaubt. Implementieren Sie die Methode `closestPoint` so, dass sie einen Punkt aus `ps` zusammen mit seiner Distanz zu `p` ermittelt, der einen minimalen Abstand zu `p` hat. Falls `ps` leer ist, wird `p` mit der Distanz 0 zurückgegeben.

c) Erstellen Sie schließlich die ***rekursive*** Methode `closestPair(cph, ps)`:
Sie ermittelt ein Punktepaar aus `ps` mit minimaler euklidischer Distanz zueinander (falls mehrere Punktepaare die gleiche minimale Distanz haben, wird ein beliebiges dieser Punktepaare zurückgegeben). Verwenden Sie dazu **unbedingt** auch die Methode `closestPoint`. Falls `ps` leer ist, soll `ClosestPairHelper.PPD_NO_RESULT` zurückgegeben werden. Falls `ps` nur einen Punkt $P(x, y)$ enthält, wird dieser als Paar mit Distanz 0 (d.h. `{x, y, x, y, 0}`) zurückgegeben. Für die Implementierung genügt ein „Brute-Force“-Ansatz, der alle Punktepaare betrachtet. Sie dürfen davon ausgehen, das kein Punkt doppelt in der Liste vorkommt.

Achtung - Wichtig: Ihre Klasse darf nur die obigen Methoden enthalten (sonst keine weiteren Methoden, Attribute, innere Klassen o.ä.) und *muss* die Methoden aus `ClosestPairHelper cph` aufrufen. *Jede einzelne* Ihrer Methoden muss gleich als erste Anweisung genau einen Aufruf der Methode `cph.traceMe();` enthalten – erfüllen Sie **unbedingt ALLE öffentlichen** Testfälle, sonst bekommen Sie für diese Aufgabe 0 Punkte! Die Verwendung von Klassen oder Methoden aus der Java-API ist **verboten**. Geben Sie Ihre Klasse `ClosestPair.java` über *EST* ab.

Gruppenaufgabe 4.4: Rekursive Kunst — 16 GP

In dieser Aufgabe sollen Sie rekursive Kunst erstellen. Dazu steht auf der Webseite mit der Klasse `Canvas` eine Leinwand sowie der Rumpf der Klasse `RekursiveKunst` für Sie bereit.
In der rekursiven Kunst hängt ein Zeichenstrich jeweils vom Strich der letzten Ebene ab. Gezeichnet wird in der Methode `draw`. Der Parameter `steps` gibt an, wie viele Ebenen noch zu zeichnen sind. Für jeden Strich mit Länge `len` sollen `branches` neue Striche mit Länge `lenFactor` · `len` auf der nächsten Ebene (von links nach rechts) gezeichnet werden. Diese sollen dabei gleichverteilt im mit `openAngle` (Bogenmaß) angegebenen Bereich sein.

a) Implementieren Sie die Methode `getNewX`. Sie berechnet aus dem x-Wert des Startpunkts, der Länge und des Winkels einer Linie den x-Wert des Endpunkts. 0 entspricht dabei der vertikalen Line nach oben (y-Wert wird größer), größere Winkel stehen im Uhrzeigersinn dazu. Hinweis: Sie dürfen dazu `Math.sin` bzw. `Math.cos` verwenden. Der Rückgabewert hat den Typ `int`, casten Sie mögliche `double`-Zwischenergebnisse so spät wie möglich!

b) Implementieren Sie die Methode `getNewY`. Sie berechnet aus dem y-Wert des Startpunkts, der Länge und des Winkels einer Linie den y-Wert des Endpunkts. Hinweis: Sie dürfen dazu `Math.sin` bzw. `Math.cos` verwenden. Der Rückgabewert hat den Typ `int`, casten Sie mögliche `double`-Zwischenergebnisse so spät wie möglich!

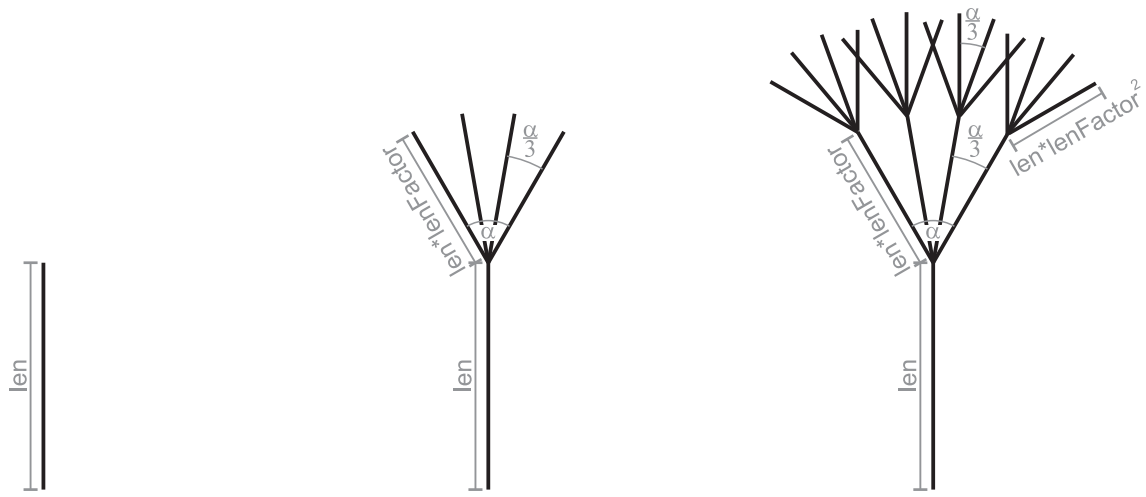


Abbildung 1: Rekursive Kunst in drei Schritten vom einzelnen Strich der Länge `len` bis zu 16 Strichen der Länge `len`$\cdot$ `lenFactor`^{2} auf der dritten Ebene. α entspricht dabei `openAngle`. Bei 4 `branches` pro neuer Ebene ergibt sich ein Winkel von $\frac{\alpha}{3}$ zwischen den einzelnen neuen Strichen.

c) Implementieren Sie die Berechnung der Länge einer Linie in der nächsten Ebene in der Methode `getNewLen`. Die Linien werden dabei jeweils abhängig von `lenFactor` verkleinert (siehe Abb. 1).

d) Implementieren Sie die Berechnung des Winkels einer Linie auf der nächsten Ebene in der Methode `getNewAngle`. Dabei gibt `startAngle` den Winkel der Linie am linken Rand, `angleDiff` den Unterschied zwischen zwei Linien an. `branch` nummeriert die Linien (von links nach rechts, 0-indiziert) durch (siehe Abb. 1).

e) Implementieren Sie das rekursive Zeichnen in der Methode `draw`. Zeichnen Sie zuerst (sofern notwendig - Abbruchfall!) einen Strich mit der Methode `drawLine` der Klasse `Canvas`, d.h. mittels `canvas.drawLine(...)`. Berechnen Sie sich dazu mit Startpunkt, Winkel und Länge den Endpunkt der Linie (mittels `getNewX`/`getNewY`). Vor den rekursiven Aufrufen muss für jeden Strich die neue Länge bzw. der neue Winkel berechnet werden (mittels `getNewLen`/`getNewAngle`). Berechnen Sie dazu zuerst den linken Rand bzw. `startAngle` und `angleDiff` (siehe Abb. 1).

f) Die Zeichnung ist noch zu regelmässig und soll zufälliger werden (vgl. Abb. 3). Ändern Sie dazu die Implementierung von `getNewAngle`, so dass der berechnete Winkel um β schwankt. Ändern Sie ebenfalls die Methode `getNewLen`, die die Variation des Längenfaktors mit einbeziehen soll (siehe Abb. 2). Um eine zufällige Abweichung im Bereich $\pm\beta/2$ bzw. $\pm r/2$ zu berechnen, verwenden Sie `getNextRandomAngle` bzw. `getNextRandomLen`.

Geben Sie Ihre Lösung als `RekursiveKunst.java` über EST ab.

25 EP + 35 GP = 60 Punkte

Abbildung 2: Die Abweichung des Winkels der neuen Striche ist um $\pm\beta/2$ erlaubt, wobei β `angleRandomness` entspricht. Die Länge der neuen Striche soll im Intervall $[$`len`$\cdot($`lenFactor` $-\frac{r}{2})$;`len`$\cdot($`lenFactor` $+\frac{r}{2})[$ liegen, wobei r `lenRandomness` entspricht.

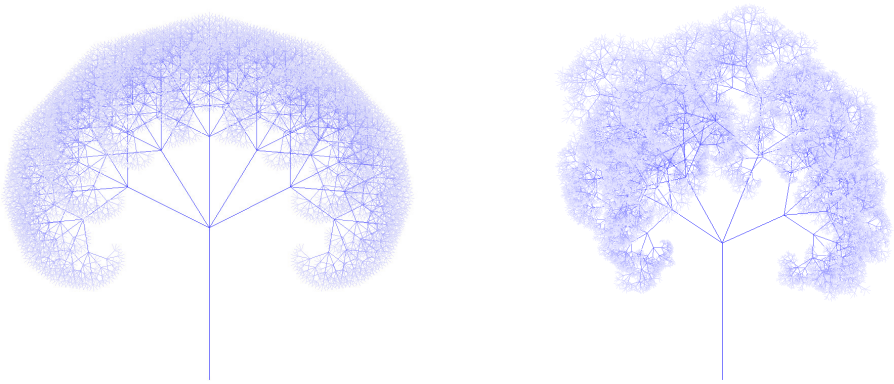


Abbildung 3: Zwei rekursive Gemälde: das linke ohne Zufall mit 7 Ebenen, 5 `branches` pro Ebene, `lenFactor`$=0.6$, `openAngle`$=0.7$, `x`$=500$, `y`$=10$ und `len`$=300$; das rechte Bild mit Zufall mit 9 Ebenen, 4 `branches`, `lenFactor`$=0.6$, `lenRandomness`$=0.35$, `openAngle`$=0.7\cdot\pi$, `angleRandomness`$=\pi/6$, `x`$=500$, `y`$=10$ und `len`$=270$.