Übungen zu Algorithmen und Datenstrukturen für Medizintechnik
Sommersemester 2021
René Raab, Robert Richer

Lehrstuhl für Maschinelles Lernen und Datenanalytik
Department Artificial Intelligence in Biomedical Engineering (AIBE)
(https://www.mad.tf.fau.de)

Aufgabenblatt 6 vom 06. Juni 2021, Abgabe am 20. Juni 2021, 22:00 Uhr

---

Achtung: Wir korrigieren die Programmieraufgaben auf diesem Blatt komplett manuell. Sie bekommen also vom EST kein Feedback dazu, ob Ihre Abgabe kompiliert!

Aufgabe 6.1: Komplexität und Aufwandsabschätzung

Punkte siehe StudOn
Komplexität

Aufgabenstellung und Abgabe (individuell, nicht als Gruppe!) im StudOn.

Aufgabe 6.2: Fragen zu Rekursion

Punkte siehe StudOn
Rekursion

Aufgabenstellung und Abgabe (individuell, nicht als Gruppe!) im StudOn.

Aufgabe 6.3: Die Geschichte vom Farbeimer 7 Punkte

Rekursion

Alice hat ein Paint-Programm geschrieben, mit dem man am Computer mit der Maus malen kann. Allerdings kann man mit Alice' Programm nur einzelne Pixel ausmalen. Da Alice gerne auch mit einem Klick komplette geschlossene Flächen in einer Farbe füllen möchte, bittet Sie Bob um Hilfe. Bob hat vor kurzem Programmieren gelernt und versichert Alice, dass er das hinbekommt. Nach einigen verzweifelten Stunden des Nachdenkens und Ausprobierens, stellt Bob fest: »Das geht nicht. Man kann keinen Algorithmus schreiben, der ganze Flächen ausmalt.« Carol bekommt das mit und entgegnet: »Doch, das geht ganz einfach. Mit Rekursion! Du musst Dir nur überlegen wie du das Füllen von Flächen in einen Basisfall und ein kleineres Problem zerlegen kannst. Der Rest passiert quasi von ganz alleine!«

In dieser Aufgabe sollen Sie den rekursiven Algorithmus zum Füllen von Flächen implementieren, auf den sich Carol bezogen hat. Ein weißer Pixel gehört genau dann zu einer geschlossenen Fläche, wenn er mindestens einen direkten Nachbarn (horizontal oder vertikal) hat, der zur Fläche gehört. Ein schwarzer oder blauer Pixel gehört nie zu einer auszufüllenden Fläche.

1. Kopieren Sie die im StudOn bereitgestellten Klassen `Paint` und `PaintCan` in ein neues Projekt. Bearbeiten Sie für diese Aufgabe nur die Klasse `PaintCan`. Die Methode `fillBob` enthält den Code, den Bob bereitgestellt hat. Wenn Sie das Programm ausführen, können Sie bereits Bobs Algorithmus ausprobieren. Das grafische Interface hat oben zwei Auswahlfelder. Beim ersten können Sie den zu verwendenden Algorithmus auswählen (entweder Bobs Algorithmus oder den rekursiven Algorithmus). Beim zweiten Auswahlfeld können Sie ein Bild auswählen, das Sie ausmalen möchten.

   Mit der linken Maustaste können Sie selber ein Bild malen, indem Sie einzelne Pixel anklicken oder mit der Maus klicken und ziehen. Mit der rechten Maustaste können Sie eine Fläche mit dem gewählten Algorithmus ausfüllen lassen. Falls Sie eine Maus mit einer mittleren Maustaste haben, können Sie diese benutzen, um Pixel zu löschen (also weiß zu färben).

   Bobs Versuch eines Algorithmus zum Füllen von Flächen haben wir Ihnen in der Methode `fillBob(int x, int y)` mitgegeben. Dieser Algorithmus funktioniert nicht.

2. Beschreiben Sie in einem Kommentar, warum Bobs Methode nicht jede Fläche vollständig füllt. *(2 Punkte)*

3. Die Methode `fillRec(int x, int y)` soll die Fläche, die den Punkt (x, y) enthält, rekursiv füllen. Überlegen Sie sich, wie Sie das Problem in ein kleineres Problem und einen Basisfall zerlegen können. Beschreiben Sie diese Überlegungen in einem Kommentar. *(2 Punkte)*

4. Implementieren Sie die rekursive Methode `fillRec` basierend auf Ihren Überlegungen. *(3 Punkte)*

5. Testen Sie Ihre Implementierung mit verschiedenen Bildern.

6. **Zusatzfrage (keine (Bonus-)Punkte):** Falls Sie AuD-MT bereits einmal gehört haben, überlegen Sie, wie das Füllen einer Fläche mit dem Traversieren von Graphen zusammenhängt. Schreiben Sie Ihre Überlegungen in einen Kommentar.

Ihre rekursive Lösung funktioniert jetzt (hoffentlich) für kleine und mittelgroße Flächen. Wenn man allerdings versucht eine sehr große Fläche (z.B. den Hintergrund oder das leere Bild) auszumalen, gibt es einen StackOverflowError. Rekursion ist also keine allgemeine Lösung für alle Probleme. Wie die Geschichte mit Alice, Bob und Carol weitergeht und wie Dave das Problem mit dem StackOverflowError löst, sehen wir auf dem Aufgabenblatt zu Graphen.

Aufgabe 6.4: Grundlagenübung Rekursion 12 Punkte

Rekursion

1. Legen Sie ein neues Java-Projekt `06-Recursion` an. Erstellen Sie eine neue Klasse `Recursion` mit einer `main`-Methode.

2. Schreiben Sie eine rekursive Methode `public static void countdown(int n)`, die die Zahlen von n bis 0 in *absteigender* Reihenfolge auf die Standardausgabe ausgibt. n ist eine natürliche Zahl größer als 0.

   Überlegen Sie sich:

   - Welche Zahl muss in `countdown(n)` ausgegeben werden?
   - Was muss passieren, wenn man `countdown(n-1)` aufruft?
   - Was ist eine geeignete Abbruchbedingung?

   Implementieren Sie nun die Methode `countdown(int)` und testen Sie sie aus der `main`-Methode!

3. Wenn die Methode `countdown(int)` richtig funktioniert, kopieren Sie sie und benennen Sie die Kopie um in `countup(int)`. Denken Sie auch daran, den Namen des rekursiven Aufrufs anzupassen (so, dass innerhalb von `countup(int)` nur `countup` und nicht `countdown` aufgerufen wird).

   Verschieben Sie jetzt **eine** Zeile an eine andere Stelle, so dass ein Aufruf von `countup(n)` die Zahlen von 0 bis `n` in *aufsteigender* Reihenfolge ausgibt!

4. Implementieren Sie nun die Methode `int hornerRecursive` mit den Parametern `int[] digits` (1. Parameter) und `int base` (2. Parameter) zur rekursiven Berechnung der Dezimalzahl z aus $(b_{n-1}b_{n-2}\dots b_1b_0)_B$ gemäß des Horner-Schemas. Die Ziffern b_j werden dabei in einem Array `digits` der Länge n gemäß $\{b_{n-1},\dots b_1, b_0\}$ verwaltet. Hierbei ist `base` die Basis B des Ursprungs-Zahlensystems. Implementieren Sie die Methode **in-place**, d.h. ohne Verwendung zusätzlicher Arrays außer `int[] digits`.

   Hinweis: Für Ihre Lösung kann es hilfreich sein, neben `hornerRecursive` eine zusätzliche Hilfsmethode zu implementieren und diese zu verwenden.

   Testen Sie Ihre rekursive Implementierung in der `main`-Methode. Testen Sie insbesondere Sonderfälle beim Aufruf Ihrer Methode.

5. Nun beschäftigen wir uns im Folgenden mit der rekursiv definierten Funktion $g(i)$:

$$g(i) = \begin{cases} 0 & \text{falls } i = 0 \\ 1 & \text{falls } i = 1 \\ 2 \cdot g(i-1) + 5 \cdot g(i-2) & \text{sonst} \end{cases} \quad (1)$$

   a) Nun soll die Berechnung von $g(i)$ auf unterschiedliche Arten in Java umgesetzt werden. Implementieren Sie dafür in der Klasse `Recursion` die folgenden Methoden:

      i. Die Methode `long gNaive(int i)` soll $g(i)$ wie in Formel 1 dargestellt mit einer kaskadenartigen Rekursion berechnen. Dies ist die einfachste, aber auch langsamste Berechnungsmethode.

      ii. Die Berechnung wird deutlich schneller, wenn man sie in eine Rumpfrekursion überführt. Um $g(i)$ zu bestimmen, benötigt man nur die Werte von $g(i-1)$ und $g(i-2)$. Diese werden – zusätzlich zu `int i` – als Parameter `long giMinusOne` und `long giMinusTwo` zur rekursiven Hilfsfunktion `gTailHelper` aufgenommen und in rekursiven Aufrufen "durchgereicht". Die Methode `long gTail(int i)` soll `gTailHelper` dann mit geeigneten Parametern aufrufen, um $g(i)$ zu berechnen.

      iii. Abschließend soll $g(i)$ in `gIter(int i)` iterativ (nicht rekursiv!) berechnet werden. Beginnen Sie dabei bei $g(0)$ und $g(1)$ und rechnen Sie zu $g(i)$ hoch.

   Hinweis: Sie können davon ausgehen, dass keine Ihrer Methoden mit $i < 0$ aufgerufen wird.

   Tipp: Um die Laufzeiten zu vergleichen, können Sie z.B. $g(46)$ berechnen. `gNaive` kann dafür > 10 Sekunden brauchen, `gTail` und `gIter` sollten fast keine Zeit beanspruchen. Die aktuelle Systemzeit als `long` bekommen Sie mittels `System.currentTimeMillis()`.

6. Geben Sie die Datei `Recursion.java` mit allen von Ihnen implementierten Methoden ab!

Aufgabe 6.5: Sierpinski-Dreieck 11 Punkte

Rekursive Kunst

Das Sierpinski-Dreieck ist ein Fraktal, das auf den polnischen Mathematiker Wacław Sierpiński zurückgeht. Abbildung 1 zeigt ein Beispiel für so ein Dreieck.

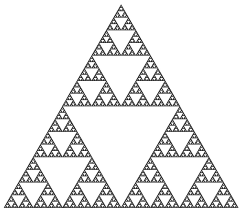

Abbildung 1: Sierpinski-Dreieck

Es handelt sich um einen rekursiven Prozess: Aus einem Dreieck wird in der Mitte ein umgedrehtes Dreieck entfernt. Aus den verbleibenden drei Dreiecken wird wieder jeweils ein Dreieck in der Mitte entfernt und so weiter. Abbildung 2 zeigt die ersten Stufen eines solchen Dreiecks, beginnend mit einem vollständigen Dreieck.

Abbildung 2: Rekursionsstufen des Sierpinski-Dreiecks von Stufe 0 bis Stufe 4.

Ihre Aufgabe wird es diesmal sein, ein solches Dreieck rekursiv zu zeichnen. Zudem wollen wir sogenannte *Event-Handler* implementieren, damit wir die Rekursionstiefe und die Farbe des Dreiecks mittels Tastatureingaben verändern können. Dazu haben wir Ihnen im StudOn die abstrakte Klasse `SierpinskiTriangleAbstract.java` für das Grundgerüst bereitgestellt. In der davon abgeleiteten Klasse `SierpinskiTriangle.java` sollen Sie dann die abstrakten Methoden implementieren.

1. Legen Sie ein neues Projekt `06-SierpinskiTriangle` an.

2. Erstellen Sie die Klasse `SierpinskiTriangle` und lassen Sie sie von `SierpinskiTriangleAbstract` erben. Lassen Sie Eclipse alle abstrakten Methoden überschreiben.

   Graphics und Color
   Informieren Sie sich in der Java-Dokumentation im Internet über die Klassen `java.awt.Graphics` und `java.awt.Color`!
   Beide Klassen wurden bereits in der Oberklasse importiert, Sie können Sie also direkt verwenden. In der Oberklasse ist ebenfalls bereits ein `Graphics`-Objekt `g` angelegt, welches Sie zum Zeichnen benötigen werden.

3. Implementieren Sie zunächst die Methode `drawTriangleRec`. Diese ist für das rekursive Zeichnen des Dreiecks verantwortlich:

   a) Schauen Sie sich Abbildung 3 an. Bei einem Aufruf von `drawTriangleRec` erhalten Sie die Koordinaten der Punkte A, B und C. Berechnen Sie zuerst die Breite (also die Länge der Grundseite) und die Höhe des Dreiecks.

   b) Berechnen Sie nun aus den Koordinaten für A, B und C und der Breite sowie der Höhe die Koordinaten der Punkte D, E und F.

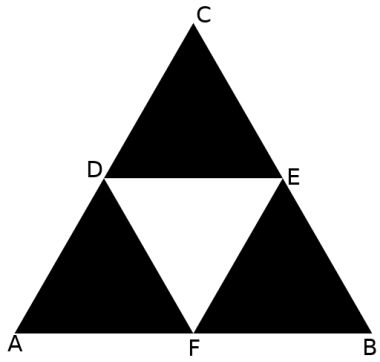

Abbildung 3: Wichtige Punkte im Dreieck.

Kleiner Tipp: Um die y-Koordinaten zu erhalten, müssen Sie die Höhe des Dreiecks halbieren. Für die x-Koordinaten müssen Sie die Breite des Dreiecks in vier gleich große Teile teilen (also vierteln).

c) Zeichnen Sie nun ein Dreieck ABC in der Farbe `color`. »Löschen« Sie nun das mittlere Dreieck DFE, indem Sie ein weißes Dreieck zeichnen.

Beachten Sie, dass `java.awt.Graphics` keine Methode speziell zum Zeichnen von Dreiecken anbietet und Sie daher `fillPolygon(int[] xPoints, int[] yPoints, int nPoints)` benutzen müssen. Ein Dreieck ist eigentlich nur ein spezielles Polygon mit drei Punkten.

d) Nun sollen die verbleibenden schwarzen Dreiecke rekursiv behandelt werden. Rufen Sie dazu `drawTriangleRec` für jedes der Dreiecke mit den korrekten Punkten (siehe Abbildung 3) auf.

e) Neue Dreiecke sollen nicht mehr gezeichnet werden, falls sie die Seitenlänge 2 unterschreiten würden oder die Rekursionstiefe 0 ($\rightsquigarrow$ `depth`) erreicht ist. Fügen Sie am Anfang der Methode die entsprechenden Abbruchbedingungen ein. Beachten Sie allerdings, dass wir bei einer Rekursionstiefe von 0 immer noch ein schwarzes Dreieck sehen möchten.

4. Wenn Sie nun in der `main`-Methode eine Instanz Ihres Programms aufrufen, sollte nun das rekursiv erstellte Dreieck wie in Abbildung 4 erscheinen.

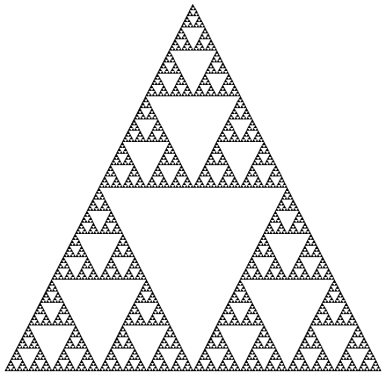

Abbildung 4: Sierpinski-Dreieck der Stufe 7 (ohne Fensterrahmen).

5. Kümmern wir uns nun darum, die Event-Handler für unser Dreieck zu implementieren. Dazu steht die Methode `handleInput(int keyCode)` bereit, die als Parameter den Tasten-Code einer gedrückten Taste übergeben bekommt. Die verschiedenen Tasten-Codes sind als öffentliche, statische Integer-Konstanten in der Klasse `java.awt.event.KeyEvent` definiert. Implementieren Sie in dieser Methode die folgenden Funktionalitäten und aktualisieren Sie die Zeichenfläche am Ende der Methode mittels `paint(getGraphics());`:

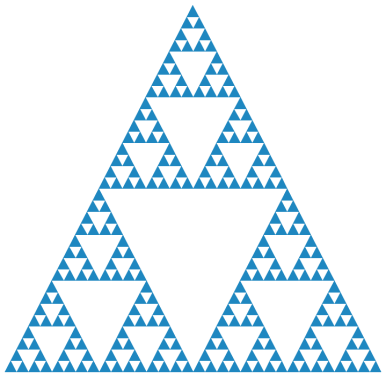

Abbildung 5: Sierpinski-Dreieck der Stufe 5 mit Zufallsfarbe.

- Das Programm soll auf das Drücken der Pfeil-oben- und Pfeil-unten-Tasten auf Ihrer Tastatur reagieren, um die Stufe, bis zu der das Sierpinski-Dreieck gezeichnet werden soll, zu verändern. Die Rekursionstiefe wird durch das Attribut `protected int depth` in der Klasse `SierpinskiTriangleAbstract` festgelegt.
- Verwenden Sie die Konstanten `KeyEvent.VK_UP` und `KeyEvent.VK_DOWN`, um die Rekursionstiefe bei Drücken der ↑-Taste um 1 zu erhöhen und beim Drücken der ↓-Taste um 1 zu erniedrigen.
- Stellen Sie sicher, dass `depth` am Ende der Methode innerhalb des Wertebereiches $[0, 7]$ liegt. Legen Sie die minimal bzw. maximal mögliche Rekursionstiefe als Konstanten `MIN_DEPTH` bzw. `MAX_DEPTH` an.
- Beim Drücken der Leertaste ($\rightsquigarrow$ `KeyEvent.VK_SPACE`) soll der Zufallsfarben-Modus ein- bzw. ausgeschaltet (»getogglet«) werden. Der Farbmodus wird durch das Attribut `protected boolean useRandomColor` der Oberklasse festgelegt. Verändern Sie das Attribut in der Methode `protected void toggleRandomColor()` und rufen Sie die Methode in `handleInput(int keyCode)` auf.
- Um das Dreieck nun farbig werden zu lassen, müssen Sie zu guter Letzt die Methode `protected void drawTriangle()` der Oberklasse überschreiben. Unterscheiden Sie in dieser Methode, ob das Dreieck mit einer Zufallsfarbe gezeichnet werden soll oder in schwarz. Erzeugen Sie im ersten Fall ein `Color`-Objekt zufälliger Farbe, im zweiten Fall ein `Color`-Objekt schwarzer Farbe und weisen Sie dieses Objekt dem Attribut `protected Color color` der Oberklasse zu.

  Achtung: Vergessen Sie nicht, am Ende die Methode der Oberklasse aufzurufen, um das Zeichnen des Dreiecks auszuführen!

6. Ihr Programm sollte nun – inklusive Event-Handler – funktionieren. Beim Aktivieren des Zufallsfarben-Modus sollte das Sierpinski-Dreieck wie in Abbildung 5 aussehen.
7. Geben Sie die Datei `SierpinskiTriangle.java` im EST ab.

Sollte Ihr Programm nicht übersetz- bzw. ausführbar sein, wird die Lösung mit 0 Punkten bewertet. Stellen Sie also sicher, dass IntelliJ IDEA keine Fehler in Ihrem Programm anzeigt, Ihr Programm übersetz- und ausführbar ist sowie die in der Aufgabenstellung vorgegebenen Namen und Schnittstellen *exakt* eingehalten werden. Geben Sie am Schluss die Dateien `PaintCan.java`, `Recursion.java` und `SierpinskiTriangle.java` über die EST-Webseite ab. Wenn Sie die Aufgabe zusammen mit einem Übungspartner bearbeitet haben, geben Sie im EST unbedingt dessen Gruppenabgabe-Code an! Kontrollieren Sie, ob Ihre Namen am Anfang aller Dateien angegeben sind – schreiben Sie im Quellcode Ihre Angaben in einen Kommentar. Im EST-Abgabesystem können Sie modifizierte Dateien mehrfach abgeben. Nur die zuletzt hochgeladene Version wird bewertet.