Übung 3 — Abgabe 15.5.2019

Aufgabe 1 – Unittests — 20 Punkte

Informieren Sie sich über das Python-Modul "`unittest`" (docs.python.org/3/library/unittest.html – wichtig sind insbesondere die Abschnitte 26.4.1, 26.4.4 und 26.4.8.1). Benutzen Sie die Funktionalität aus diesem Modul, um Tests für Sortierfunktionen zu implementieren. Das Grundgerüst der Lösung finden Sie im File `sorttest.py` auf Moodle. Geben Sie die vervollständigte Datei ab. Um ein Array zu kopieren benutzen Sie `deepcopy()` aus dem Modul `copy`.

a) Warum beginnen in `sorttest.py` einige Funktionennamen mit `test`..., andere jedoch mit `check`…? Beantworten Sie die Frage in einem Kommentar in der Datei. *1 Punkt*

b) Ein Sortieralgorithmus für Arrays muss drei Nachbedingungen erfüllen: die Arrays müssen vor und nach dem Sortieren die gleiche Größe haben, die gleichen Elemente enthalten, und das Ergebnis muss sortiert sein. Implementieren Sie in der Funktion `checkIntegerSorting()` Tests, die diese Bedingungen für Arrays von ganzen Zahlen prüfen. Nutzen Sie in geschickter Weise Memberfunktionen der Klasse `list` und Testfunktionen wie `assertEqual()`. Denken Sie dabei daran, dass Zahlen mehrfach vorkommen können: [3,2,3,1] → [1,1,2,3] ist zwar sortiert, enthält aber nicht mehr die gleichen Elemente. *5 Punkte*

c) Implementieren Sie den Integer-Teil der Funktionen `testBuiltinSort()` (testet Pythons Sortierfunktion) und `testInsertionSort()` (testet das in der Datei gegebene `insertionSort()`), indem Sie nach dem Sortieren `checkIntegerSorting()` aufrufen. Führen Sie die Tests mit dem Kommandozeilenaufruf '`python sorttest.py`' aus und vergewissern Sie sich, dass beide Sortieralgorithmen korrekt arbeiten. *3 Punkte* *4 Punkte*

d) Implementieren Sie nun die Funktion `checkStudentSorting()`. Sie soll zunächst mit denselben Bedingungen wie in b) testen, dass ein Array von Studenten nach der Klausurnote (also nach der Membervariable '`mark`') sortiert ist. **Fügen Sie danach Tests dafür hinzu, dass die Sortierung stabil ist.** Diese Tests müssen allgemein implementiert sein, d.h. sie dürfen keine Annahmen darüber machen, wie die Arrayelemente vor der Sortierung angeordnet waren.

e) Implementieren Sie den Studenten-Teil von `testBuiltinSort()` und `testInsertionSort()` und führen Sie die Tests aus. Um nach der Klausurnote zu sortieren, müssen Sie die Sortierfunktionen mit dem Parameter `key=Student.getMark` aufrufen. Es sollte herauskommen, dass Pythons Sortieralgorithmus stabil ist, `insertionSort()` hingegen nicht (d.h. der Test schlägt fehl). *2 Punkte* *2 Punkte*

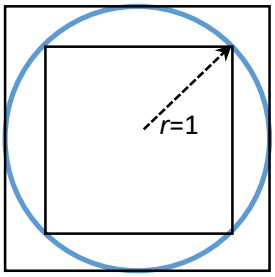
f) Finden und korrigieren Sie den Fehler in `insertionSort()`, so dass alle Tests funktionieren.

g) Implementieren Sie `mergeSort()` und fügen Sie die Funktion `testMergeSort()` mit den gleichen Tests wie bei den anderen Verfahren hinzu. Im Unterschied zur Vorlesung muss die Funktion `mergeSort()` jetzt den 'key'-Parameter unterstützen. Denken Sie daran, dass `mergeSort()` nicht in-place arbeitet. *3 Punkte*

Aufgabe 2 – Implementieren und Debuggen des Algorithmus von Archimedes zur Bestimmung von π — 17 Punkte

Die Korrektheit numerischer Algorithmen ist besonders heikel, weil solche Algorithmen auch dann falsche Resultate liefern können, wenn sie eigentlich mathematisch korrekt implementiert wurden. Das liegt daran, dass der Datentyp `float` und die dafür verfügbaren arithmetischen und algebraischen Funktionen nur Näherungen der exakten mathematischen Ausdrücke liefern können. Die Näherungen sind so gut, dass man normalerweise keine unangenehmen Überraschungen erlebt, aber diese Übungsaufgabe demonstriert, dass Fehler bereits bei einfachen Aufgaben auftreten können.

Gute Schätzwerte für π wurden bereits in der Antike benötigt (z.B. um die Größe und damit den Preis eines Grundstücks mit gekrümmten Grenzen zu bestimmen). Archimedes (ca. 287 – 212 v. Chr.) hat dafür eine geniale Methode erfunden: Man zeichne zunächst einen Einheitskreis (Radius $r = 1$), dessen Umfang nach Definition 2π ist. Dann konstruiere man ein regelmäßiges *n*-Eck, das den Kreis genau von innen berührt, und eines, das den Kreis genau von außen berührt. Die Skizze zeigt dies für Quadrate ($n = 4$). Das innere *n*-Eck hat stets einen kleineren Umfang als der Kreis, das äußere einen größeren. Wenn s_n und t_n die Seitenlängen des inneren und des äußeren n-Ecks sind, gilt somit



$$n\,s_n < 2\pi < n\,t_n \quad \text{bzw.} \quad \frac{n}{2}\,s_n < \pi < \frac{n}{2}\,t_n$$

Im Falle des Quadrats findet man leicht $s_4 = \sqrt{2}$ und $t_4 = 2$, also $2.82 < \pi < 4$. Diese Schätzung ist natürlich noch ziemlich grob. Kennt man jedoch die Seitenlänge der n-Ecke für ein bestimmtes n, kann man die Seitenlängen für $2n$ einfach nach folgenden Formeln berechnen:

$$s_{2n} = \sqrt{2 - \sqrt{4 - s_n^2}} \quad \text{und} \quad t_{2n} = \frac{2}{t_n}\left(\sqrt{4 + t_n^2} - 2\right) \tag{1}$$

Durch wiederholtes Verdoppeln von n bekommt man immer bessere Schätzwerte, weil die n-Ecke den Kreis immer besser approximieren. Archimedes hat seinerzeit mit dem Sechseck begonnen und gelangte nach vier Verdoppelungen zum 96-Eck. Sein Ergebnis $\frac{223}{71} < \pi < \frac{22}{7} \approx 3{,}142$ (mit nur 0.04% Abweichung) war viele Jahrhunderte lang der beste bekannte Schätzwert. Erst 500 Jahre später wurden in China 5 Dezimalstellen berechnet, und erst vor 400 Jahren kehrte der Rekord nach Europa zurück, nachdem Ludolph van Ceulen (1540-1610) sich über 30 Jahre lang abgemüht hatte, um 60 Verdoppelungen auszurechnen, natürlich per Hand. In dieser Übung wollen wir mit Quadraten beginnen und dann 30 Verdoppelungen ausführen (wir haben danach 4 Gigaecken!).

a) Implementieren Sie den Algorithmus als Funktion `archimedes1(k)` (wobei k die Anzahl der Verdoppelungen ist) und geben Sie für jedes n die Zahl der Ecken, den unteren und den oberen Schätzwert für π sowie deren Differenz aus. Der Code für alle Aufgabenteile soll im File „`archimedes.py`“ abgegeben werden. Fragen beantworten Sie in Kommentaren in diesem File. *2 Punkte*

b) Wenn Sie den Algorithmus korrekt implementiert haben, sind die Ergebnisse zunächst sehr vielversprechend, aber ab einem gewissen n geht etwas schief, und es kommen immer unsinnigere Werte heraus. Beschreiben Sie Ihre Beobachtungen und finden Sie den Grund für dieses Verhalten. Lesen Sie dazu den WikiPedia-Artikel zum Stichwort Auslöschung und schauen Sie sich die Teilterme der Formeln genau an. *3 Punkte*

c) Numerikexperten empfehlen, die Formeln (1) für die Verdoppelung von n durch folgende Formeln zu ersetzen *6 Punkte*

$$s_{2n} = \frac{s_n}{\sqrt{2 + \sqrt{4 - s_n^2}}} \quad \text{und} \quad t_{2n} = \frac{2\,t_n}{\sqrt{4 + t_n^2} + 2} \tag{2}$$

Zeigen Sie (z.B. mit Hilfe der binomischen Formeln), dass die neuen Formeln (2) mathematisch äquivalent zu den alten Formeln (1) sind. Implementieren Sie den Algorithmus mit den neuen Formeln als Funktion `archimedes2(k)` und überzeugen Sie sich, dass das Verfahren jetzt funktioniert. Warum sind die neuen Formeln besser? Wie viele zusätzliche Dezimalstellen von π bekommt man in etwa pro Verdoppelung?

d) Ludolph van Ceulen hätte seine Arbeit bereits nach 15 Jahren beenden können, wenn ihm aufgefallen wäre, dass man die Verdoppelungen für das äußere n-Eck gar nicht ausrechnen muss. Statt dessen kann man t_n einfach am Schluss aus s_n berechnen: *6 Punkte*

$$t_n = \frac{2\,s_n}{\sqrt{4 - s_n^2}}$$

Leiten Sie diese Beziehung her (z.B. mit dem Satz des Pythagoras) und implementieren Sie damit einen Test, der die Ergebnisse Ihrer Funktion `archimedes2(k)` verifiziert, aber bei

`archimedes1(k)` einen Fehler signalisiert. Bei welchem n schlägt Ihr Test zuerst Alarm? (Vielleicht hat van Ceulen etwas ähnliches gemacht, denn ihm ist trotz der langen Rechnung kein Fehler unterlaufen, ganz im Gegensatz zu William Shanks, der 1873 sogar 707 Dezimalstellen von π veröffentlichte, aber ab Position 528 waren seine Ergebnisse leider falsch.)

Bonusaufgabe (10 Punkte): Leiten Sie die Formeln (1) für die Berechnung der Seitenlängen her. Eine ausführlich kommentierte englische Übersetzung der originalen Herleitung von Archimedes kann man übrigens im Internet nachlesen: ab Seite 93 unter der URL http://www.aproged.pt/biblioteca/worksofarchimede.pdf.

Bitte laden Sie Ihre Lösung bis zum 15.5.2017 um 12:00 Uhr auf Moodle hoch.