Übung 3 Abgabe 10.5.2008

Aufgabe 1 – Debuggen durch manuelles Nachvollziehen 4 Punkte

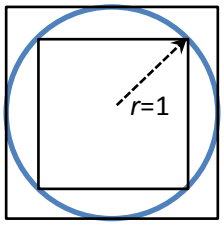
a) In der Vorlesung haben wir die Division durch sukzessives Subtrahieren behandelt, um Vorbedingungen, Nachbedingungen und Invarianten zu erläutern und einen formalen Korrektheitsbeweis exemplarisch vorzuführen:

```
Vorbedingungen: x > 0, y > 0
q = 0
r = x
while y <= r:
    r = r - y
    q = q + 1
Nachbedingungen: y > r
Invarianten:     xᵢ == x and yᵢ == y and x == rᵢ + y * qᵢ
```

(*q* ist der Quotient, *r* der Rest von *x* / *y*, das Subskript *i* bezeichnet die Werte der Variablen nach dem *i*-ten Durchlauf durch die Schleife). Erstellen Sie per Hand eine Tabelle, die die Werte der Variablen nach dem *i*-ten Schritt auflistet und die Invarianten überprüft. Eine solche Tabelle ist eine langweilige, aber sehr effektive Methode zum Debuggen von Algorithmen, weil man sofort sieht, wo der Algorithmus den ersten Fehler macht:

Schritt	x	y	q	r	r + y*q
0	31	9	0	31	31+9*0 == 31
1	...				
...					

Aufgabe 2 – Implementieren und Debuggen des Algorithmus von Archimedes zur Bestimmung von π 12 Punkte



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 $2.82 < \pi < 4$. Das sind natürlich ziemlich ungenaue Schätzwerte. Kennt man jedoch die Seitenlänge der n-Ecke für ein bestimmtes n, kann man die Seitenlängen für $2n$ einfach nach folgenden Formeln berechnen:

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

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.

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 (Sie müssen dazu die Teilterme der Formeln genau anschauen).

c) Numerikexperten empfehlen, die Formeln für die Verdoppelung von n durch folgende Formeln zu ersetzen

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

Zeigen Sie (z.B. mit Hilfe der binomischen Formeln), dass die neuen Formeln mathematisch äquivalent zu den alten 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:

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

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 (8 Punkte): Leiten Sie die ursprünglichen Formeln für die Berechnung der Seitenlängen her. Eine ausführlich kommentierte englische Übersetzung der originalen Herleitung von Archimedes kann man übrigens im Internet nachlesen: ab PDF-Seite 281 unter der URL ia700307.us.archive.org/31/items/worksofarchimede029517mbp/worksofarchimede029517mbp.pdf.

Aufgabe 3 – Unittests 8 Punkte

Informieren Sie sich über das Python-Modul "unittest" (docs.python.org/library/unittest.html – wichtig sind insbesondere die Abschnitte 25.3.1 und 25.3.4). Implementieren Sie die Funktion `checkSorting()` aus Übung 2.1 als Unit Test. Benutzen Sie die Funktionen `assert_()`, `assertEqual()` usw. um die einzelnen Bedingungen zu überprüfen. Geben Sie die Lösung im File "`sorttest.py`" ab.

Aufgabe 4 – Double-ended Queue (Deque) 16 Punkte

Eine double-ended Queue (Deque) vereinigt die Fähigkeiten einer Queue (first in – first out) mit denen eines Stacks (last in – first out). Wenn die Deque eine feste Maximalgröße hat, kann sie mit Hilfe eines Arrays implementiert werden, das als *Ringspeicher* arbeitet: Die Funktion `push()` füllt das

Array von vorn nach hinten, die Funktionen `popFirst()` und `popLast()` leeren es wieder in der Queue- bzw. Stackreihenfolge. Der gerade aktive Bereich des Arrays wird durch einen Anfangs- und einen Endindex bezeichnet. Wenn diese Indizes beim Einfügen oder Auslesen die Arraygrenzen überschreiten, werden sie zyklisch behandelt (d.h. max+1 $\rightarrow$ 0 und $0 - 1 \rightarrow$ max) – dies realisiert den "Ring". Natürlich darf der Endindex nie den Anfangsindex "überrunden". Die Größe des Ringspeichers wird beim Konstruktoraufruf festgelegt.

a) Geben Sie Vor- und Nachbedingungen für die Funktionen `q=Deque(N)`, `q.size()`, `q.capacity()`, `q.push(x)`, `x=q.popFirst()` und `x=q.popLast()` vollständig an (dabei ist N die Maximalgröße, `q.capacity()` gibt die Maximalgröße zurück [diese ist fest und kann nicht verändert werden], `q.size()` ist die aktuelle Größe, `x` das einzufügende bzw. ausgelesene Element).

b) Implementieren Sie die Datenstruktur in Python. Achten Sie dabei darauf, dass jede Funktion ihre Vorbedingungen prüft und eine Exception vom Typ `RuntimeError` auslöst, falls diese nicht erfüllt sind. Die Fehlermeldung sollte klar ausdrücken, was der Nutzer tun muss, um den Fehler zu vermeiden.

c) Implementieren Sie mit dem unittest-Modul geeignete Unit Tests für Ihre Datenstruktur. Testen Sie dabei die Nachbedingungen im typischen Fall (teilweise gefüllte Queue), die Randfälle (leere Queue, fast volle Queue), sowie die Fehlerfälle (d.h. ob jeweils die erwarteten Exceptions ausgelöst werden, wenn Vorbedingungen nicht erfüllt sind). Achten Sie darauf, dass sämtliche Teile des Codes getestet werden ("complete code coverage").

   Implementation und Tests sollen im File "`deque.py`" abgegeben werden.