Übung 3 Abgabe 2.5.2008

Aufgabe 1 – Korrektheit 40 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: *4 Punkte*

```
Vorbedingungen: x > 0, y > 0
q = 0
r = x
while y <= r:
    r = r - y
    q = q + 1
Nachbedingungen: y > r
Invarianten:     x_i == x and y_i == y and x == r_i + y * q_i
```

(*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):

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

b) Wir haben gezeigt, dass sich der Algorithmus von Freivalds zum Testen der Matrixmultiplikation eignet, weil die Wahrscheinlichkeit, einen vorhandenen Fehler trotz wiederholter Anwendung des Tests nicht zu entdecken, exponentiell (mit $1/2^K$) sinkt. Der Algorithmus lautet: *12 Punkte*

```
Eingabe: NxN-Matrizen A, B, C, von denen behauptet wird, dass C = A*B
         Anzahl K der Wiederholungen
1.  Für i = 0...K-1:
    a. wähle einen Vektor u der Länge N mit zufälligen Nullen und Einsen
    b. berechne die Vektoren v = A*(B*u) und w = C*u (beachte die Klammern!)
    c. falls v != w: return False      # Fehler gefunden
2.  return True                        # keinen Fehler gefunden
```

Implementieren Sie die Matrixmultiplikation und den Algorithmus von Freivalds.[3] Zur Erzeugung von Zufallszahlen verwenden Sie das Python-Modul "random" (siehe docs.python.org/lib/module-random.html). Bauen Sie jetzt absichtlich Fehler in die Matrixmultiplikation ein (verändern Sie ein zufälliges Element von A oder C, brechen Sie eine Schleife einen Schritt zu früh ab, und dergleichen – probieren Sie mehrere Fehlerarten aus und beschreiben Sie, was Sie gemacht haben) und lassen Sie den Testalgorithmus für verschiedene *K* laufen. Erstellen Sie eine Tabelle oder ein Diagramm, wie häufig der Fehler für jedes *K nicht* gefunden wurde. Wird der erwartete exponentielle Abfall der Häufigkeiten bestätigt?

c) Informieren Sie sich über das Python-Modul "unittest" (docs.python.org/lib/module-unittest.html – wichtig sind insbesondere die Abschnitte 23.3, 23.3.1 und 23.3.5). 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. *6 Punkte*

[3] Matrizen sollen hier durch eindimensionale Arrays implementiert werden, so dass z.B. das Matrixelement C_{ij} als `C[i+j*N]` adressiert wird.

d) 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→0, $0 - 1 \rightarrow$max – dies realisiert den "Ring"). Natürlich darf der Endindex nie den Anfangsindex "überrunden". *18 Punkte*

I. 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, `q.size()` ist die aktuelle Größe, x das einzufügende bzw. ausgelesene Element).

II. Implementieren Sie die Datenstruktur in Python (siehe untenstehende Kurzanleitung zu Python-Klassen). 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.

III. 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.

Kurzeinführung in Python-Klassen

Klassen werden in Python durch das Schlüsselwort "`class`" deklariert. Dahinter folgen eingerückt die Methoden, die diese Klasse unterstützen soll, jeweils beginnend mit dem Schlüsselwort "`def`". Der Konstruktor muss dabei den Namen "`__init__`" haben. Die Parameterliste wird wie gewohnt gebildet, mit einer Ausnahme: der erste Parameter hat stets den Namen "`self`" und referenziert das Objekt, für das die Funktion aufgerufen wurde:

```
class Foo:
    def __init__(self, param1, param2):     # Konstruktor
        self.param1 = param1
        ...
    def bar(self):                          # weitere Methoden
        return self.param1
    ...
```

Basisklassen werden hinter dem Klassennamen in Klammern angegeben, die Konstruktoren der Basisklassen werden im Konstruktor explizit aufgerufen:

```
class Foo(FooBase):
    def __init__(self, p1, p2):
        FooBase.__init__(self, p1, p2)     # Aufruf des Konstruktors der
                                           # Basisklasse
```

Um ein Objekt der Klasse zu erzeugen, wird der Konstruktor aufgerufen, indem der Klassenname wie ein Funktionsname verwendet wird. Danach können die Methoden des Objekts über die Punktsyntax aufgerufen werden:

```
aFoo = Foo(p1, p2)  # Konstruktor
p3   = aFoo.bar()   # Methode bar()
```