Algorithmen und Programmierung 2, SS 2016 — 2. Übungsblatt

Abgabe bis Freitag, 29. April 2016, 12:00 Uhr,

Hinweise zu den Programmieraufgaben in PYTHON: Geben Sie Ihrem Programm für Aufgabe N den Namen `Aufgabe`*N*`_`*Nachname1*`_`*Nachname2*`.py`. Beim Auruf dieses Programms aus der Kommandozeile soll eine Testsuite ablaufen, in der jede Funktion mindestens einmal aufgerufen wird. Sie dürfen Unterprogramme in getrennte Module packen, aber die Modulnamen müssen in der gleichen Weise mit Ihren Nachnamen gekennzeichnet sein. Laden Sie die Programme elektronisch im KVV hoch, und geben Sie zusätzlich die ausgedruckten Programme zusammen mit den anderen Lösungsbestandteilen in Papierform in die Fächer der Tutoren ab.

1) Verwenden Sie sprechende Namen für Variablen und Funktionen, die den Inhalt der Variablen oder die Funktionalität der Funktion charakterisieren.
2) Verwenden Sie die vorgegebenen Funktionsnamen, falls diese angegeben sind.
3) Kommentieren Sie das Programm.
4) Verwenden Sie geeignete Hilfsvariablen und Hilfsfunktionen.
5) Löschen Sie Programmzeilen und Variablen, die nicht verwendet werden.

---

10. Wörterbücher und Mengen, Sammelbilder, 0 Punkte

    (a) Untersuchen Sie experimentell, wie oft man eine zufällige Zahl zwischen 1 und n wählen muss, bis jede Zahl vorgekommen ist. Machen Sie für $n = 10$, 100, und 1000 jeweils 100 Experimente, und bestimmen Sie für jedes n das Minimum, das Maximum, und den Mittelwert der Anzahl der Zahlen. Ein Experiment besteht dabei aus der notwendigen Anzahl von Versuchen, bis man alle Zahlen beobachtet hat. (Zufällige ganze Zahlen kann man mit der Bibliotheksfunktion `random.randint` erzeugen. Der Erwartungswert für die Anzahl der Versuche ist übrigens $n(1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n})$.)

    (b) Bestimmen Sie bei jedem Experiment auch, wie oft die häufigste Zahl vorgekommen ist, und machen Sie eine analoge Statistik wie bei Aufgabe (a).

11. Boolesche Ausdrücke, 0 Punkte

    Schreiben Sie die Funktion `schaltjahr`[1] zum Testen, ob ein Jahr ein Schaltjahr ist, möglichst knapp mit einem einzigen Booleschen Ausdruck und ohne `if`-Anweisungen.

    (Ist diese Fassung leichter zu verstehen als das ursprüngliche Programm?)

12. Der erweiterte Euklidische Algorithmus, 15 Punkte

    (a) Schreiben Sie ein PYTHON-Programm `testGGT`, das die Ausgabe t, r, s des erweiterten Euklidischen Algorithmus überprüft, ob sie eine gültige Ausgabe zur Eingabe a, b ist (und ob t somit der größte gemeinsame Teiler von a und b ist).

    (b) Der größte gemeinsame Teiler t von a und b hat die Eigenschaften:

        i. t teilt a, und t teilt b, (d.h. t ist ein gemeinsamer Teiler von a und b).
        ii. Jeder gemeinsame Teiler d von a und b teilt auch t.

        Ist die ganze Zahl t für die Eingabe $a = 7$ und $b = 11$ durch diese beiden Bedingungen eindeutig charakterisiert?

    (c) Wenn man die beiden obigen Bedingungen als Richtschnur nimmt, was müsste dann $\mathrm{ggT}(a, 0)$ sein? Was müsste $\mathrm{ggT}(0, 0)$ sein?

[1] `http://www.inf.fu-berlin.de/lehre/SS16/ALP2/schaltjahr.py`

(d) Schreiben Sie eine PYTHON-Funktion zur Berechnung von $\mathrm{ggT}(a, b)$ für *beliebige* ganze Zahlen $a, b.$ mit dem erweiterten Euklidischen Algorithmus. Das Ergebnis t soll ≥ 0 sein. (Natürlich dürfen Sie nicht einfach eine Bibliotheksfunktion zu Hilfe nehmen.) Aufruf: `t,r,s = ggT(a,b)`.

13. Turm von Hanoi, 15 Punkte

Ergänzen Sie folgendes rekursive Programm[2] zur Lösung des Hanoi-Problems:

```
def Hanoi(n, Start, Hilfsstift, Ziel):
    """Turm von Hanoi

    Bewege n Scheiben, die zu Beginn auf dem Stift "Start" sind,
    zum Stift "Ziel"; "Hilfsstift" ist der dritte Stift."""
    if n==0: return
    Hanoi(n-1, Start, Ziel, Hilfsstift)
    setze_um(n, Start, Ziel)
    Hanoi(n-1, Hilfsstift, Start, Ziel)
```

(a) Verwalten Sie die Scheiben auf den Stiften, indem Sie geeignete Datenstrukturen verwenden. Diese Struktur soll eine globale Variable mit dem Namen `Stifte` sein. insbesondere soll die Prozedur `setze_um(n, x, y)` die Scheibe n von Stift x auf Stift y umsetzen, und dabei soll überprüft werden, ob die Regeln eingehalten werden.[3] Andernfalls soll das Programm eine Fehlermeldung ausgeben und mit der Anweisung `raise RuntimeError` eine Ausnahme erzeugen.

(b) Die Initialisierung `init_Hanoi(n, x, y, z)` soll die drei Stifte erzeugen und n Scheiben auf den ersten Stift setzen. Die Benennung x, y, z der Scheiben soll dabei frei wählbar sein. Ein Beispiel eines Initialisierungsaufrufs wäre:

```
init_Hanoi(10, 'A', 'B', 'C')
```

(c) Eine Funktion `anzeigen()` soll den Zustand der Stifte anzeigen.

(d) Eine Boolesche Funktion `fertig()` soll überprüfen, ob das Ziel erreicht ist.

14. Iterative Lösung des Hanoi-Problems, 0 Punkte

Beweisen Sie:

(a) Wenn n gerade ist und die Lösung mit dem Aufruf `Hanoi(n, 'A', 'B', 'C')` gestartet wird, dann bewegen sich die Scheiben mit geraden Nummern immer nur zyklisch in "aufsteigender" Richtung $A \to B \to C \to A$, und die Scheiben mit ungeraden Nummern in der umgekehrten Richtung.

(b) Es wird immer abwechselnd Scheibe 1 und eine andere Scheibe bewegt.

(c) In jeder möglichen Konfiguration gibt es höchstens drei Züge, die den Regeln entsprechen.

Mit Hilfe dieser Beobachtungen kann man die Folge der Bewegungen mit einem Programm bestimmen, das ohne Rekursion auskommt.

15. Langsamste Lösung des Hanoi-Problems, 0 Punkte, zum Tüfteln

Lösen Sie das Problem, die Scheiben von Stift A auf Stift C zu bringen, indem Sie *möglichst viele* Schritte machen, ohne dass sich jedoch eine Konfiguration wiederholt. Können Sie *alle* möglichen Konfigurationen durchlaufen? Ist die Lösung eindeutig?

[2] Siehe `http://www.inf.fu-berlin.de/lehre/SS16/ALP2/Hanoi.py`

[3] Es darf immer nur die oberste Scheibe auf einem Stift bewegt werden, und sie darf nicht auf einer kleineren Scheibe landen.