Übungen zu Computergrundlagen WS 2015/2016

Übungsblatt 10: Python II

08.01.2016

Allgemeine Hinweise

- Abgabetermin für die Lösungen ist
  - **Freitag, 15.01., 11:00**
- Schickt die Lösungen bitte per Email an Euren Tutor.

Aufgabe 10.1: Das Heronverfahren (5 Punkte)

Hat man keinen Taschenrechner zur Hand, liefert das mehr als 3000 Jahre alte Heronverfahren gute Nährungen für die reelle Wurzel. Auf speziellen Prozessoren wie Grafikkarten wird das Verfahren sogar heute noch eingesetzt, um die Genauigkeit von Wurzelberechnungen nachträglich zu erhöhen. Das Heronverfahren besteht einfach darin, die Folge

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right) \tag{1}$$

zu berechnen, die quadratisch gegen die Wurzel aus a konvergiert, d.h. mit jedem Schritt verdoppelt sich die Anzahl der signifikanten Stellen.

Das Verfahren lässt sich so verstehen: ist $x_n = \sqrt{a}$, so gilt $x_n = \frac{a}{x_n}$. Durch die Bildung des Mittelwerts aus diesen beiden Werten liegt die neue Näherung (hoffentlich) dichter an der gesuchten Wurzel.

- **10.1.1** Implementiert das Heronverfahren in Python, um die Wurzel einer beliebigen Zahl auf 5 Nachkommastellen genau zu berechnen. (3 Punkte)

  Hinweise:

  - Übersetze die mathematische rekursive Definition in eine Schleife. Warum brauchst Du den Index n nicht mehr?
  - Auch der Computer kann natürlich nicht unendlich viele Folgenglieder von Gleichung (1) berechnen. Ihr müsst Euch eine *Abbruchbedingung* überlegen, also, wann das Programm enden soll. Das ist der Fall, wenn sich die neue Näherung von der alten um weniger als die geforderte Genauigkeit unterscheidet.
  - Stellt sicher, dass das Programm auch in der Lage ist, Fehler abzufangen, also wenn man sinnlose Werte als Eingabe gibt. Testet dazu das Programm mit den Werten $a \in \{0, 2, \pi, -1\}$.
- **10.1.2** Erweitert das Programm so, dass es die Wurzeln der Zahlen von 1 bis 10 berechnet und mithilfe der `matplotlib` plottet. (1 Punkt)
- **10.1.3** Schaue dir darüber hinaus auch das Konvergenzverhalten des Heronverfahrens an. Plotte dafür den Betrag der Abweichung von der tatsächlichen Lösung gegen die Anzahl der Iterationen semilogarithmisch (y-Achse) für die Wurzel aus π. (1 Punkt)

  Hinweise:

– Plotten mit der `matplotlib`:

```
from matplotlib.pyplot import *
# Zeichnen der Funktion:
x = range(1,11)
y = []
for a in x:
    ...
        y.append(Eure Wurzelnaeherung zu a)
plot(x, y)
# semilogarithmisch:
semilogy(x, y)
# Anzeigen des plots:
show()
```

Aufgabe 10.2: Radix-Sort (5 Punkte)

Radixsort ist ein Sortierverfahren für natürliche Zahlen, dass stellenweise vorgeht (Details für Interessierte: `http://de.wikipedia.org/wiki/Radixsort`). Bei diesem Verfahren werden die Zahlen zunächst nur entsprechend der niedrigsten Ziffer sortiert. Dazu werden zehn Listen, den zehn möglichen Ziffern entsprechend, erstellt, an die die Elemente in der Reihenfolge ihres Erscheinens angehängt werden. Anschließend werden diese Listen aneinandergehängt, so dass die mit einer Null endenden Zahlen zuerst kommen, dann die mit Eins endenden usw. Nun wird mit der nächstgrößeren Ziffer fortgefahren, also 10 neue Listen gebildet, an die die Elemente angehängt werden. Dabei ist wichtig, dass in der Erscheinensreihenfolge angehängt wird, da so die Sortierung der niedrigsten Stelle erhalten bleibt. Das Verfahren endet, wenn die größte vorhandene Ziffer in der Liste erreicht ist.

Beispiel: Sei die Liste $[23, 25, 1, 20, 10]$ gegeben. Nach der ersten Ziffer sortiert, ergibt sich die neue Liste $[\underline{2}0, 1\underline{0}, \underline{1}, 2\underline{3}, \underline{5}]$, nach der zweiten, höchsten Ziffer dann [_1,$\underline{1}$0,$\underline{2}$0,$\underline{2}$3,$\underline{2}$5], also die sortierte Liste.

Implementiere den Radix-Sort Algorithmus für eine Liste L natürlicher Zahlen in Python. Euer Algorithmus benötigt folgende Teile, die Ihr *getrennt* Testen solltet:

- **10.2.1** Überlegt Euch, wie Ihr eine bestimmte Stelle einer Zahl bestimmen könnt.(1 Punkt)
- **10.2.2** Schreibt eine Schleife, die die Sortierung anhand der Ziffer für eine gegebene Stelle ausführt. Erzeugt dazu zunächste eine Liste von 10 Listen (eine für jeden möglichen Wert der Ziffer), und fügt anschließend die Elemente von L entsprechend an. Dann baut die neue, teilsortierte Liste aus diesen Teillisten zusammen, in dem Ihr sie einfach aneinanderhängt. (2 Punkte)
- **10.2.3** Bestimme das Maximum von L und dessen Anzahl an Ziffern N_L. (1 Punkt)
- **10.2.4** Setze diese Teile zu einem vollständigen Radix-Sort Algorithmus zusammen. Dazu musst Du zunächst N_L bestimmen, und dann N_L-mal die Sortierung nach einer Ziffer durchführen, die mit jeder Iteration eins weiter nach links rückt. Teste deine Implementation an einer Liste mit Zufallszahlen. (1 Punkt)

Hinweise:

- Um die n-te Stelle zu bestimmen, wirst Du die Modulo-Operation (%) benötigen.
- Erzeugen eines zufälligen Integers zwischen 1 und `MAX` in Python:

```
import random
x = random.randint(1, MAX)
```

- Nützliche Listenoperationen sind `append` und das Zusammenführen mittels +.