Unit 16

Dr. Günter Kolousek

21. Juli 2015

Lege wiederum ein Verzeichnis an. Nennes es `28_unit16`! In diesem Verzeichnis sollen alle Dateien der jeweiligen Einheit abgelegt werden.

1 Schulübungen

Hier beschäftigen wir uns jetzt mit dem Sortieren von Sequenzen und beginnen mit Bekanntem:

1. Schreibe eine Funktion `sort1(a, b, c)`, die 3 Zahlen als Parameter bekommt und ein (aufsteigend) sortiertes Tupel dieser Zahlen zurückliefert.

   Beispiel:

   ```
   >>> sort1(4, 2, 3)
   (2, 3, 4)
   ```

   Tipp: Verwende eine `if`-Anweisung mit mehreren `elif`. Denke daran, dass in Python Vergleichsoperatoren eigentlich nicht streng binär sind, d.h. `a < b < c` funktioniert.

   Frage: Wie ist die Funktion umzuschreiben, sodass ein absteigend sortiertes Tupel zurückgeliefert wird?

2. Schreibe nun eine Funktion `sort2(seq)`, die eine Sequenz mit 3 Zahlen als Parameter bekommt und ein (aufsteigend) sortiertes Tupel dieser Zahlen zurückliefert.

   Beispiel:

   ```
   >>> sort2([4, 2, 3])
   (2, 3, 4)
   ```

   Frage: Wie ist die Funktion umzuschreiben, dass immer der gleiche Sequenztyp zurückgeliefert wird, wie der Typ des Parameters? Das sollte also z.B. folgendermaßen aussehen:

```
>>> sort2([4, 2, 3])
[2, 3, 4]
>>> sort2((4, 2, 3))
(2, 3, 4)
```

3. Implementiere nun den schlechtesten Sortieralgorithmus der Welt!

   Die Idee ist folgende: Beginne vorne. Vergleiche jeweils 2 benachbarte Zahlen und vertausche diese, wenn diese nicht in der richtigen Reihenfolge sind. Danach ist sicher die größte Zahl am rechten Ende, aber alle anderen Zahlen sind unsortiert. Beginne deshalb wieder von vorne.

   Hier ist er, der Bubble-Sort Algorithmus::

   ```
   für jedes Element der Liste:
     für jedes Element y der Liste vom Anfang bis zum vorletzten Element:
       wenn das aktuelle Element y größer ist als dessen Nachfolger:
         vertausche das aktuelle Element y mit seinem Nachfolger
   ```

   Info: Bei dem vorherigen Algorithmus wird die Laufvariable der inneren Schleife y benannt.

   Schreibe dafür eine Funktion `bubble1(lst)`, die die übergebene Liste sortiert und dann zurückliefert. Die Elemente der Liste können Zahlen, Strings, Tupel,… sein. Voraussetzung ist lediglich, dass die Elemente untereinander jeweils vergleichbar sind.

   Teste indem du die Funktion aufrufst und das Ergebnis ausgibst. Ist damit ein Beweis erbracht, dass der Algorithmus oder die implementierte Funktion ”funktioniert”?

   Warum heißt dieser Algorithmus genau so? Weil die jeweils größten Elemente wie eine Blase an die Oberfläche (an den rechten Rand kommen).

   Theorie: Warum ist dieser Algorithmus denn eigentlich so schlecht? Weil bei n Elemente größenordnungsmäßig :math:‘n^2‘ Vergleiche und Vertauschoperationen notwendig sind, da hier zwei verschachtelte Schleifen jeweils für (fast) alle Elemente durchlaufen werden.

   Kann man diesen nicht doch noch etwas verbessern? Ja, einmal durchgehen der äußeren Schleife erspart man sich auf jeden Fall! Warum? Nach dem ersten Durchlauf ist das größte Element sicher an der richtigen Stelle. Nach dem zweiten Durchlauf das zweitgrößte Element an der richtigen Stelle. Nach dem (n-1)ten Durchlauf stehen die höchsten (n-1) Zahlen an der richtigen Stelle und damit das letzte Element hier mit auch an der richtigen Stelle.

   Kann man diesen nicht noch weiter etwas verbessern? Auf zur nächsten Aufgabe.

4. Warum soll man beim zweiten Durchlauf die Liste nochmals bis zum (bitteren) Ende durchlaufen, wenn das letzte Element sicher schon in seiner richtigen Position steht? Ist doch nicht notwendig, also werden wir auch erst gar nicht so weit gehen:

   ```
   für jedes Element top der Liste vom letzten zum ersten:
    für jedes Element y vom ersten zum vorletzten Element (letztes ist top):
       wenn das aktuelle Element y größer ist als dessen Nachfolger:
         vertausche das aktuelle Element y mit seinem Nachfolger
   ```

   Info: Die erste Schleife geht hinunter. Die beste Möglichkeit hierfür ist, die Möglichkeit in Python zu nehmen, einen absteigenden Bereich zu erzeugen. Wie? Z.B.: `range(5, 2, -1)` Das vorletzte Element im Algorithmus bezieht sich auf die Schleifenvariable `top`, die das letzte Element angibt.

   Schreibe eine Funktion `bubble2(lst)`, die den verbesserten Algorithmus implementiert. Testen!

   Aber hat die Verbesserung wirklich etwas gebracht? Ausprobieren::

   ```
   import random
   import copy
   import time

   # Liste mit 1000 zufälligen Zahlen von 0 bis 1000 erstellen
   lst = [random.randint(0, 1000) for x in range(1000)]

   # 2 Kopien erstellen
   lst1 = copy.copy(lst)
   lst2 = copy.copy(lst)

   # Anfangszeit von bubble 1 messen
   t1 = time.time()
   bubble1(lst1)
   # Endzeit von bubble1 und Anfangszeit von bubble2 messen
   t2 = time.time()
   bubble2(lst2)
   # Endzeit von bubble2 messen
   t3 = time.time()

   print(t2 - t1)
   print(t3 - t2)
   ```

   Und? Wie sieht es aus?

5. Aber wie sieht das Ganze aus, wenn die Daten schon weitgehend sortiert sind, wie in `[1, 9, 2, 3, 4, 5]`? Hier würde ein Durchgang reichen und die Liste ist

schon sortiert. Hmm, wenn sich offensichtlich in einem weiteren Durchgang nichts ändert, dann kann die äußere Schleife abgebrochen werden.

Also, die Idee ist: Führe eine boolesche Variable `changed` ein, die in jedem Schleifendurchgang der äußeren Schleife auf `False` gesetzt wird. Kommt es in der inneren Schleife zu einer Vertauschung, dann setze diese auf `True`. Wird am Ende der äußeren Schleife keine Vertauschung erkannt, dann beende die äußere Schleife vorzeitig (mit `break`).

Schreibe dazu eine Funktion `bubble3(lst)`.

Teste zuerst mit der Liste `[1, 9, 2, 3, 4, 5]`. Führe dazu ein hübsches `print` zu Testzwecken in der äußeren Schleife ein, damit du siehst, wie es funktioniert. Wenn dies wie erwartet funktioniert, entferne das `print` wieder.

Erweitere in einem weiteren Schritt unser obiges Testverfahren, indem `bubble3` einerseits mit einer weiteren Kopie der Liste `lst` gemessen wird und danach mit einer vollständig aufsteigend sortierten Liste der ersten 1000 natürlichen Zahlen (`list(range(1000))`). Wie sieht es mit der Laufzeit aus?

Naja, die eine oder andere Verbesserung wäre noch möglich, aber aus dem wird nichts Vernünftiges mehr. Da muss ein anderer Algorithmus her (und als Faustregel gilt: verwende **NIE** den Bubble-Sort Algorithmus!).

6. Der Selection-Sort Algorithmus ist besser als der Bubble-Sort Algorithmus und funktioniert folgendermaßen:

   a) Lege eine neue leere Ergebnisliste an.

   b) Wähle das kleinste Element aus der Liste aus und entferne dieses aus der Liste.

   c) Hänge dieses an die Ergebnisliste an.

   d) Mache weiter bei b. bis kein Element in der Liste mehr enthalten ist.

   Dieses Verfahren heißt deshalb Selection-Sort, weil immer das kleinste Element aus der Liste ausgewählt wird.

   Hier folgt die genauere Beschreibung des Algorithmus:

```
Für jedes Element start vom ersten bis zum vorletzten:
  Belege min mit start
  Für jedes Element x von start + 1 bis zum letzten Element:
   Wenn das aktuelle Element x kleiner ist als das Element an Position min:
       Merke die Position des aktuellen Elementes x in min
 Wenn min != start, dann Element an min mit Element an start vertauschen
```

Gibt es hier eine Ergebnisliste wie in der verbalen Beschreibung erklärt? Ja? Nein? Warum?

Schreibe eine Funktion `selection(lst)`, die den Selection-Sort Algorithmus implementiert.

Teste wiederum die Laufzeiten analog zu dem Testen der Laufzeit der Bubble-Sort Varianten!

Laufzeiten gemessen auf einem Acer AS1810TZ Subnotebook. Bei mir kam es zu folgender Ausgabe auf einem Subnotebook:

```
bubble1: 0.617578983307s
bubble2: 0.384479999542s
bubble3: 0.401995897293s
bubble3/sortiert: 0.000426054000854s
selection: 0.178674936295s
selection/sortiert: 0.176262140274s
```

Da sieht man, dass der Bubble-Sort Algorithmus mehr als drei Mal der Zeit benötigt als der Selection-Sort Algorithmus.

Achtung: Diese Laufzeiten hängen ab von: dem Rechner (hauptsächlich CPU, Speicher, aktuelle Auslastung) und den aktuellen Daten (zufällige Zahlen!).