TPOP PRACTICAL

FUNCTIONS, LISTS, DICTIONARIES & FILES

PRACTICAL 4 – Additional Problems

Problem I: *Text analysis*

We want to write a program to analyse a text document. We want to be able to do the following statistics:

1. Word frequency, e.g. how many times a word appears in the text.
2. Number of words in the text.
3. Average length of words.
4. How many words of length *n* appear in the text?

You should assume that the document is a very large document taken from a file (see hints below). It would be too costly to open\read the file every time we want to extract one statistic. Ideally we would like to read the file only once, store the necessary information in an appropriate data structure, and then be able to extract the required statistics efficiently.

To test your algorithm, and before implementing reading text files, you could copy and paste the variable `sampleText` from the file `practical_4_part1.py`.

Hints:

Below is a sample code to read a line from a file. To open a file use the built-in function `open`, use `help(open)` to know more. To read a line from a opened file use the `readline()` method. To know more about file use `help(file)`, to know more about the `readline` method use `help(file.readline)`. Note that the `readline` retains the newline `'\n'` in the returned string.

```
Sample code: Read from a text file

>>> file_name = input('Enter the file name: ')
Enter the file name: test.txt
>>> my_file = open(file_name, "r")
>>> line = my_file.readline()
>>> line
'this is a sample text in a file\n'
>>> print "First line from file is: ", line
First line from file is: this is a sample text in a file
>>> my_file.close()
```

Problem II: *Sparse Matrices (Very difficult)*

In the subfield of numerical analysis, a **sparse matrix** is a matrix populated primarily with zeros. If the majority of elements differ from zero then it is common to refer to the matrix as a **dense matrix**.

$$S = \begin{bmatrix} 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix}$$

The native data structure for a matrix is a two-dimensional array. Each entry in the array represents an element $a_{i,j}$ of the matrix and can be accessed by the two indices i and j. For an $m \times n$ matrix, enough memory to store up to $(m \cdot n)$ entries to represent the matrix is needed.

In our example S contains only 6 non-zeros elements. Using the representation used in the last practical we need to store 36 values. Using what we know so far about Python, how could you store a sparse matrix more efficiently? Try to find the solution by yourself first, then as a group. If after 10 minutes you are still stuck, have a look at Wikipedia. If still stuck ask me (NOT a demonstrator) for help.

Once you have found a data structure, write three functions:

1. A function taking a sparse matrix as a parameter, and returns the dimensions of that matrix. For S it should return `(6, 6)`. (relatively easy)
2. A function taking two sparse matrices as parameters, and returns the sum of the two matrices. Remember the constraints on the matrices. (bit more difficult)
3. A function taking a sparse matrix as a parameter, and returns the transpose of that matrix.

If you want to pull your hair out, try the multiplication!