COMP 1405Z
Fall 2022 – Tutorial #4

Objectives

- Practice writing code using multi-dimensional lists
- Practice using lists and dictionaries to perform text analysis

Problem 1 – Matrix Multiplication

As part of the course project, you will be required to implement the PageRank algorithm. This will involve several matrix and vector operations. In this problem, you will create a module to perform some of these operations. You will then be able to use this module within your project.

Within this problem, matrices will be represented as 2D lists. Each element in the first dimension of these lists will represent a row of the matrix. Each index in the second dimension will represent a column within the matrix, and each entry in the second dimension will represent the entry at that particular row/column within the matrix. To demonstrate, an example matrix and its list representation is included below:

Matrix	List Representation
$A = \begin{bmatrix} 9 & 3 & 1 \\ 2 & 2 & 1 \\ 4 & 1 & 6 \end{bmatrix}$	A = [[9, 3, 1], [2, 2, 1], [4, 1, 6]]

To start this problem, open the provided matmult.py file, which has the function signatures for the three functions you will have to implement. The first function, **`mult_scalar(matrix, scale)`** must return a new 2D list containing the result when the given matrix is multiplied by the given scalar (i.e., a given integer/float value). **The original matrix passed as an argument to the function must not be modified**. If you are unfamiliar with matrix multiplication operations, multiplying by a scalar is simply a matter of multiplying each value within the matrix by the given scalar value. So, if we multiplied the example above by the scalar 3, we would get:

```
A = [ [9, 3, 1],                mult_scalar(A, 3) = [ [27, 9, 3],
      [2, 2, 1],                                      [6, 6, 3],
      [4, 1, 6] ]                                     [12, 3, 18] ]
```

The second function you must implement is the **`mult_matrix(a, b)`** function. This function should return a new matrix that is the result of multiplying the matrix **`a`** by the matrix **`b`**. The

original matrices must not be modified. The rules for multiplying a matrix by a matrix are slightly more complex than scalar matrix multiplication. You can find a step-by-step outline of the process at https://mathsisfun.com/algebra/matrix-multiplying.html. The challenge will be producing an algorithm based on this description that uses list operations. If the inputs are invalid (i.e., matrices do not have compatible dimensions), you can return the value None.

The third function you must implement is the **`euclidean_dist(a,b)`** function. This function accepts two single-row matrices (i.e., vectors) using the list representation for matrices used so far in this problem (e.g., a = [[9, 3, 1]]). Your function must calculate the Euclidean distance between these two vectors. To calculate the Euclidean distance between two *n*-dimension vectors, you can use the equation below (note: this is very similar to calculating distance between two points, but written in a general way to support higher dimensions):

$$euclidean\_dist(a,b) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \cdots + (a_i - b_i)^2 + \cdots + (a_n - b_n)^2}$$

Two test files are included in the tutorial zip file for checking the correctness of your functions. matmult-tester-short.py will call each function 10 times and stop if any case does not produce the correct output. matmult-tester-full.py will perform a similar process but will run each function 10000 times with different inputs.

Problem 2 - Text Analysis

Create a Python file called **analysis.py** that will perform text analysis on files. The goal of this module is to load a file once and allow multiple queries to be executed on the data from that file without reading the file again. For this problem, assume that each space (" ") in the document separates one word from the next. You can also assume there is no punctuation or other symbols present in the files – only words separated by spaces. If you want to see examples of the type of text, look in the testfile_.txt files included in the tutorial zip file.

You must implement and test the following functions inside of your analysis.py file:

1) load(str) – Takes a single string argument, representing a filename. The program must open the file and parse the text inside. This function should initialize the variables (e.g., lists, dictionaries, other variables) you will use to store the information necessary to solve the remainder of the problems. These variables should be created outside the function so that they can be accessed by the other functions you will define. **This way, the file contents can be parsed once and the functions below can be executed many times without re-reading the file, which is a relatively slow process**. This function should also remove any information stored from a previous file when it is called. That is, when the load function is called, all information about the previously loaded file is forgotten.
2) commonword(list) – Takes a single list-type argument which contains string values. The function should operate as follows:
    a. If the list is empty or none of the words specified in the list occur in the text that has been loaded, the function should return None.

   b. Otherwise, the function should return the word contained in the list that occurs most often in the loaded text - or any one of the most common, in the case of a tie.
3) commonpair(str) – Takes a single string argument, representing the first word. This function should return the word that most frequently followed the given argument word (or one of, in case of ties). If the argument word does not appear in the text at all, or is never followed by another word (i.e., is the last word in the file), this function should return None.
4) countall() – Returns the total number of words found in the text that has been loaded. That is, the word count of the document.
5) countunique() – Returns the number of *unique* words in the text that has been loaded. This is different than the previous function, as it should not count the same word more than once.

You can use the analysistester.py file from the tutorial zip, along with the included testfileX.txt files, to test your functions.

Submission

Add each of your code files and the testing resources to a single .zip file named "**tutorial4.zip**" and submit it to Brightspace. Ensure that your file is a .zip and has the proper name. Download your submission afterward and check the contents to ensure everything works as expected.