Course:	**INFO6143**
Professor:	**Jim Cooper**
Project:	**Lab #4**
Due Date:	**Thursday, March 7, 2024 11:30 pm**
Submitting:	**Submit your zipped solution to the Lab 4 submission folder**

How will my lab be marked?

Marks Available	What are the Marks Awarded For?	Mark Assigned
1	Load the dictionary from the wiki-news-300d-1M.vec file	
1	Word entry loop (3 words) works according to description below	
1	Maintain the six word vectors necessary to find the best analogy word	
1	3 included screen captures of correct output for known word vector analogies	
1	Python file is **zipped** and submitted on-time to the submission folder	
5	Total	

Lab Description

Create a single Python program using the editor/IDE of your choice. Name this source file using the following template: your first name followed by an underscore and then "Lab4". For example, "Jim_Lab4.py".

For this lab, we'll only have a single requirement and as such we'll just create a Python app that can produce a duplicate of the sample output (see the console screen capture near the end of this description).

Overview

This lab is a continuation of Lab #3 and will give you a chance to create a Python console application that does word vector analogies. By this we mean:

man is to woman as king is to queen
man is to woman as prince is to princess
boy is to girl as man is to woman
bad is to good as sad is to happy
doctor is to hospital as teacher is to school
grass is to green as sky is to blue
Rome is to Italy as Athens is to Greece
Canada is to Ottawa as Australia Canberra

Not all analogies you think of will work, however any of the ones listed above will work so long as you use the full Wikipedia database from the Facebook Fast Text site: "wiki-news-300d-1M.vec"

https://fasttext.cc/docs/en/english-vectors.html

The only problem with the full "million" word database, is that it will be slower to load and process, because of the size. This makes debugging difficult and so I recommend using the smaller version we introduced with Lab #3 (FastText100K.txt), until you have most of the initial debugging done.

Also, grading everyone's submissions for even a subset of the analogies that work will be overly time consuming, using the complete database. As a result, we want you to include your name in the output, as illustrated with the examples at the end of this doc. **When you're ready to submit your lab, complete three valid analogy runs and take a screen capture of the console for each one**.

When I'm grading your lab, I'll select one of the analogies and run it for testing purposes.

Detailed Steps for Doing Lab #4

Since you already have a version of the "word vector analogies" applications in a different programming language (C#), part of the goal for doing this project will be to extract useful information from already existing and working code and reuse the information in the construction of your own Python solution.

Since I've already compared the output from the C# version of the word vector analogies application to my own Python version, I know that you can get (mostly) duplicate results from both solutions, assuming you use the same word vector data file. I've included another C# version on the FOL site for anyone who doesn't already have the app from the data science course.

Begin by loading your word vector file using your relevant code from Lab #3.

The user input must consist of three separate words (your keyboard input code must check for three words and end the application if a string of zero words is entered or force the user to redo if the number of words is > 0 but not equal to three).

When you have your list of three words, retrieve from the dictionary the word vector for each of the three words. The easiest way to do this is to create three separate list vectors:

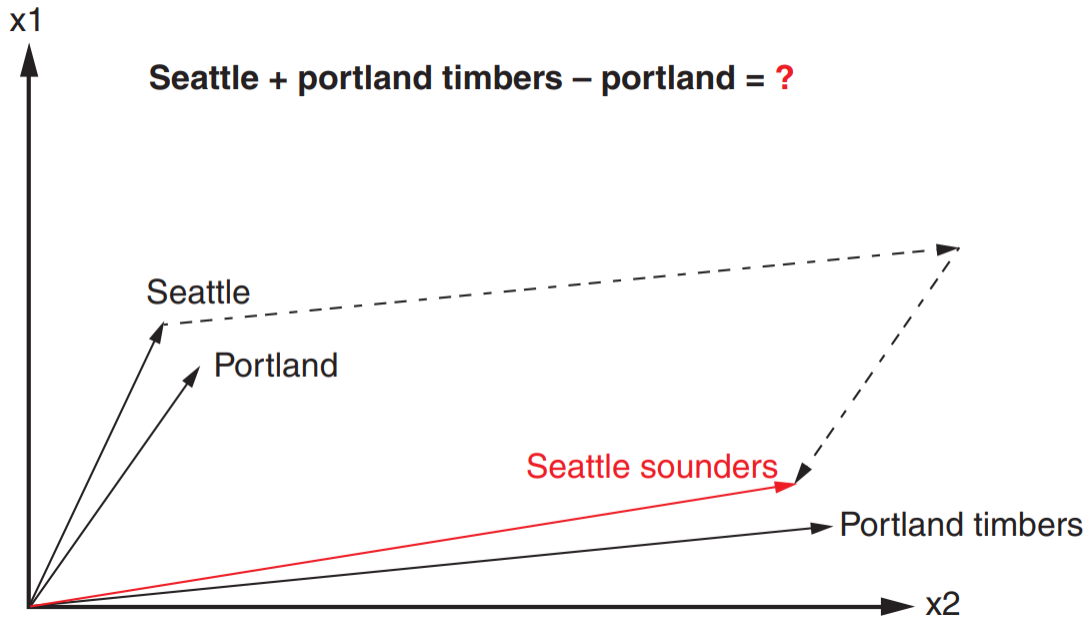
vector1, vector2 and vector3

Next create a fourth vector (vector4) by subtracting your vector1 from vector2.

Now create a fifth vector (vector5) by adding vector4 and vector3.

Adding or subtracting vectors means adding (or subtracting) each number in the vector from(to) the corresponding number in the other vector. You will end up with a new vector that has the same number of elements as the original two vectors, in our case, always 300.

You've already had a look at some of these ideas in your data science course, so we won't spend too much time redoing the basic concepts. I do, however, have a diagram that illustrates the concept of word vectors in terms of geometry:



FYI, the Sounders and Timbers are soccer teams for the cities of Seattle and Portland.

In this example, Seattle and Portland represent two lines (vectors) that are in relative proximity. In addition, we have two other lines, Seattle sounders and Portland timbers which are also in relative proximity. By adding Seattle and Portland timbers and subtracting Portland, we can get a value that's close to Seattle sounders.

This example, while it's geometrically correct, doesn't work with our word vector dataset because despite the words being in the dataset, they probably don't exist within the right context, i.e., professional soccer teams.

Word vector analogies tend to get better and better as the amount of word vector data increases, however matches are never guaranteed.

As with Lab #3, we now go through the entire dictionary and take the word vector (vector6) for each word in the dataset and use vector5 (above) to get the cosine similarity (for vector5 and vector6).

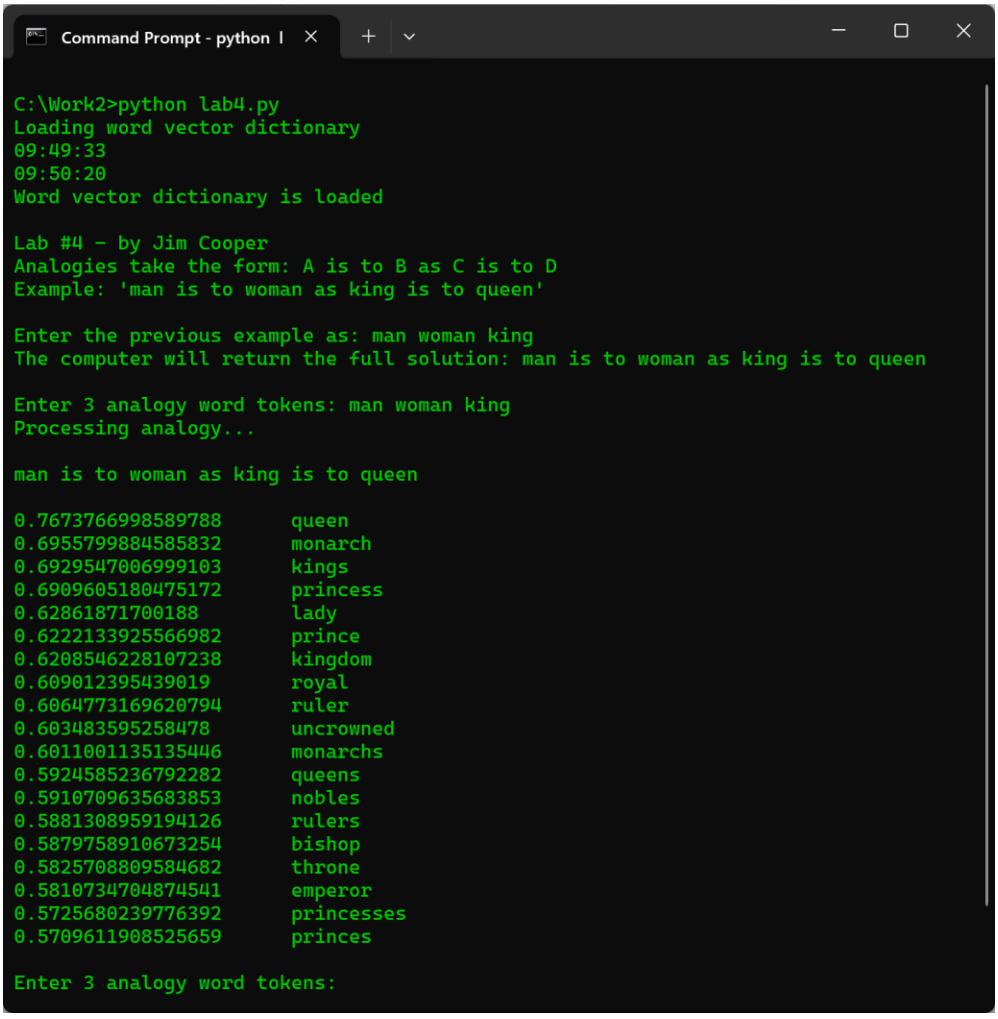
For this lab, we're asking you to do something slightly more complicated when it comes to determining the sequencing of your output. In Lab3 most people compared the cosine similarity number to the current highest and if the new one is higher, it becomes the current highest value.

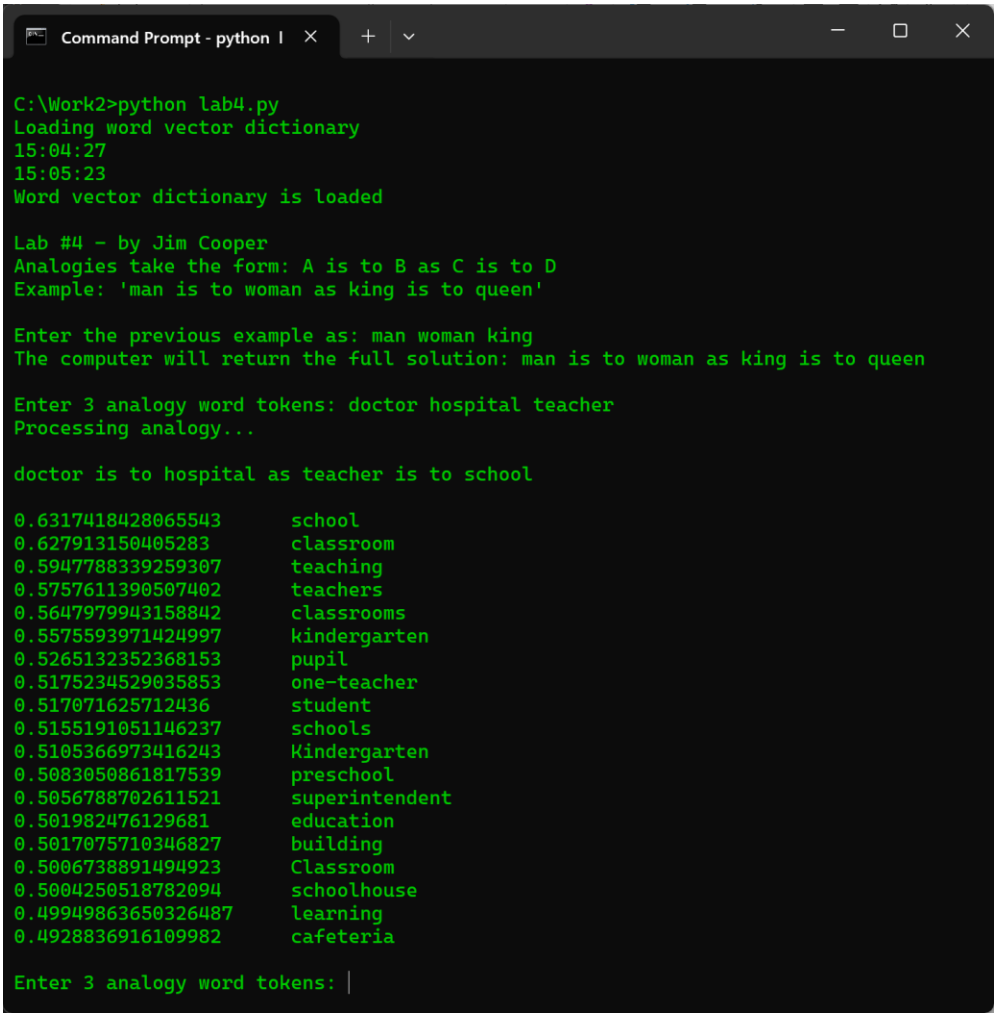
For Lab4, we want you to keep track of the top 20 cosine similarity scores in a more precise way. As you traverse the dictionary and calculate new cosine_similarity values, compare the value to **each** of the numbers in your top 20 list and if it's larger than any of the existing values, include it in the list and remove the bottom value.

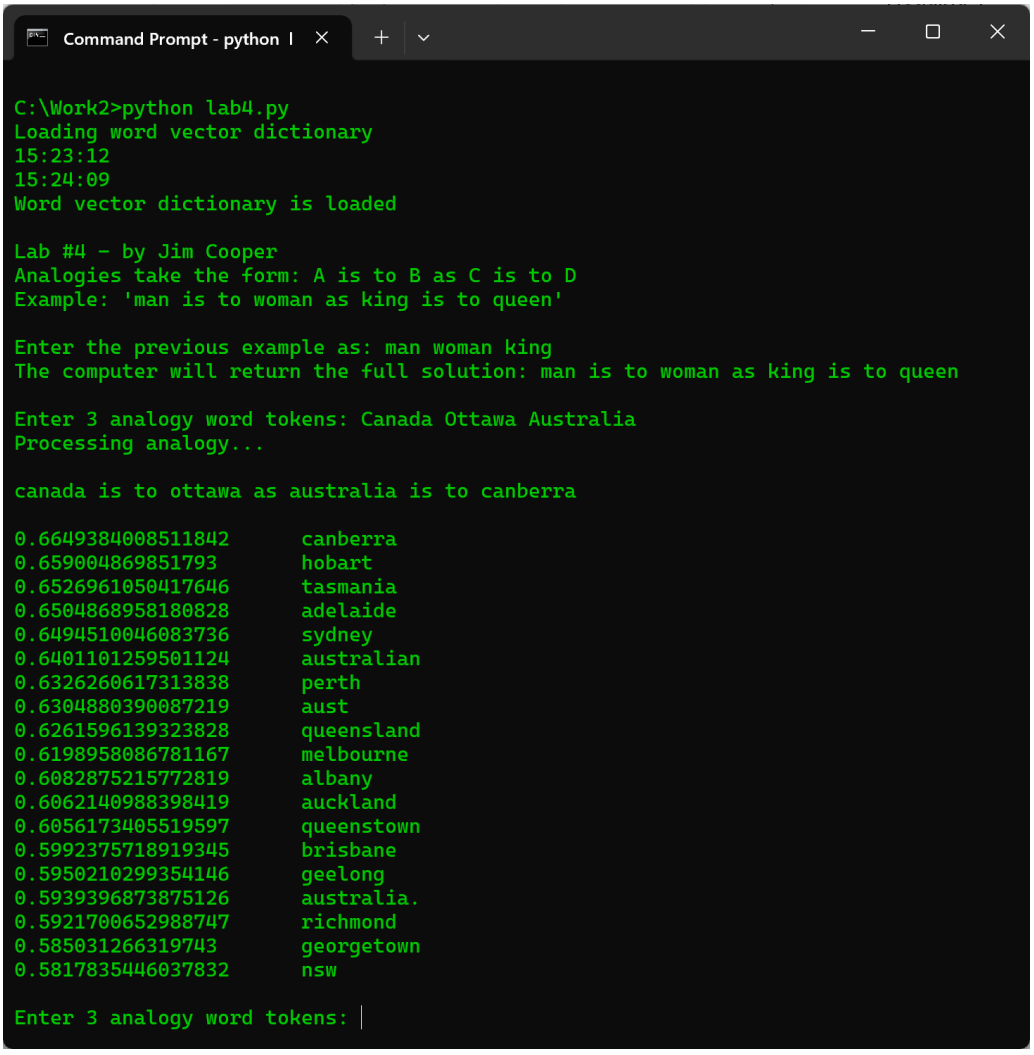
This sounds like a lot of work, but if you preset your list size to 20 (filled initially with 0.0 values) and then subsequently, if you determine the next cosine_similarity value derived from the dictionary should be included, just replace the last item in the list (it will always be the smallest) and re-sort the list from the top down.

I've left out some of the details related to formatting the output with the cosine_similarity value followed by the word but that should be easy enough to do. As you can see from the sample output, I've inserted a tab character between the value and the word.

Here's are some samples for this lab:




Command Prompt - python l
C:\Work2>python lab4.py
Loading word vector dictionary
15:04:27
15:05:23
Word vector dictionary is loaded
Lab #4 - by Jim Cooper
Analogies take the form: A is to B as C is to D
Example: 'man is to woman as king is to queen'
Enter the previous example as: man woman king
The computer will return the full solution: man is to woman as king is to queen
Enter 3 analogy word tokens: doctor hospital teacher
Processing analogy...
doctor is to hospital as teacher is to school
0.6317418428065543 school
0.627913150405283 classroom
0.5947788339259307 teaching
0.5757611390507402 teachers
0.5647979943158842 classrooms
0.5575593971424997 kindergarten
0.5265132352368153 pupil
0.5175234529035853 one-teacher
0.517071625712436 student
0.5155191051146237 schools
0.5105366973416243 Kindergarten
0.5083050861817539 preschool
0.5056788702611521 superintendent
0.501982476129681 education
0.5017075710346827 building
0.5006738891494923 Classroom
0.5004250518782094 schoolhouse
0.49949863650326487 learning
0.4928836916109982 cafeteria
Enter 3 analogy word tokens:



Finally:

Submit your **zipped** Python source code and screen captures to the Lab 4 submission folder.