COMP 200 & COMP 130
Assignment 3: Regular Expressions, Dictionaries, Text Analysis, Edit Distance

Be sure to read the course policies, as posted on the course web site:

http://www.clear.rice.edu/comp200/policies.shtml

Work in assigned pairs on this assignment. Each pair should put all assignment answers in one Python file called *netid1_netid2_3.py*, substituting in the students' NetIDs. Put the answers to non-programming problems in comments. When submitting, only one of the member should turn in the pair's answers.

Total points: 100 for COMP 200, 150 for COMP 130.

Wherever reasonable, use functions that have been defined previously on this assignment, on a previous assignment, or in class.

Regular Expressions – COMP 200 & COMP 130 (25 points)

1. (20 points total – 5 points each)

   In each of the following, we will describe the syntax of words that are of interest. We will also indicate how many "hits" there are for such words in the British National Corpus. You will give (in Python comments) the following two items:

   - The corresponding input to the British National Corpus' regular expression search at

     http://sara.natcorp.ox.ac.uk/

     You can check your own answers against the expected number of hits.

     See the BNC's description of their regular expression syntax:

     http://www.natcorp.ox.ac.uk/tools/chapter4.xml?ID=FIMNU#PAQ

   - The string pattern for `re.match` that will match against exactly those words.

   These are very similar to the examples in our notes.

   (a) The word, or more properly abbreviation, "uu". (10 hits)

   (b) Words such as "unusual" and "tumultuous" containing three or more "u"s. (19252 hits)

   (c) Words such as "vacuum" that contain two or more consecutive "u"s. (2089 hits)

   (d) Words, abbreviations, and Roman numerals that contain three or more consecutive, identical vowels. (7656 hits)

2. (5 points)

   Briefly describe the differences between the behaviors of `re.search` and `re.match`.

Dictionaries – COMP 200 & COMP 130 (75 points)

Statistics

3. (20 points)

   Write a function `modes` that computes the *modes* of a list. A mode is the most commonly occurring value in the input. Since multiple values may occur the same number of times, there may be multiple modes.

   Two illustrative examples:

   - `modes([5,1,9,3,9,4])` should return `[9]`. It returns a list of the only mode, not just a number.
   - `modes(["cat","dog","duck","dog","cat")` should return `["cat","dog"]` or `["dog","cat"]`. The modes may be listed in any order.

   Your code should start by building a dictionary of the occurrence counts.

   You are not allowed to use a Python Counter, since it has this feature already defined.

Text Analysis

In the notes found on

http://www.clear.rice.edu/comp200/12spring/notes/14/text-analysis1.shtml

we provided code that would take a filename and produce a dictionary (or Counter) of the word and punctuation counts for that file's contents. Also, previously, you've written functions to compute various statistics. We are going to combine these ideas to provide very simple analyses of the documents.

In the following, we will use the term "word" to include punctuation symbols, abbreviations, and the like, that are returned by the provided code.

4. (20 points total – 5 points each part)

   Save the file containing Shakespeare's complete works

   http://www.clear.rice.edu/comp200/resources/texts/Shakespeare Complete Works.txt

   to your compute. Put it in the same directory as your code.

   Answer the following questions for the text file. Provide not only the value requested (in a Python comment), but the code you wrote to find it.

   (a) Count the number of distinct words.

   (b) Find the average (i.e., mean) word frequency.

   (c) Find the median word frequency. As before in the course, use the lower median.

   (d) What percentage of the total number of words do "the" and "The" together represent?

   Since knowing whether your answers are correct or not is difficult, we strongly recommend you first test your code on the small text files, where you can also compute the desired results by hand.

Simulation

When driving many urban highways, you can see signs that tell you the time needed for traffic to reach some destination. In the following problems, you will write a version of these signs.

On the highway, we will use two sensors: one at the beginning of the timed area and one at the end. As a car passes the first sensor, a car's highway toll pass (such as Houston's EZPass) or an occupant's cell phone is detected. The unique identifier of the toll pass or cell phone is recorded, along with the current time. At the second sensor, the car or phone is again detected, and again its ID and time are again recorded.

The highway sign only displays the traffic information part of the time, since the sign is also used for other purposes. When the sign is displaying traffic information, it requests the current average transit time every few minutes. This is when all the calculations occur.

We want the posted average transit time to be up to date, so we throw away data that is sufficiently old. As an added benefit, that also ensures that we don't continually need more storage to save all the data.

5. (35 points)

   We will use a globally-defined dictionary `enteringTimes` for each car's first-sensor time and another globally-defined dictionary `leavingTimes` for each second-sensor time.

   For simplicity, the times will just be numbers, the number of hours since midnight. Thus, 4pm is the number 16.

   Write three functions that will work together to simulate this timing. Each function has an argument that indicates the current time.

   - `sensor1(carID,time)` – Adds a sensor reading to `enteringTimes`.
   - `sensor2(carID,time)` – Adds a sensor reading to `leavingTimes`.
   - `update(time)` – Returns the average elapsed time and removes old data from the two dictionaries. More specifically, it deletes the sensor readings from `enteringTimes` that are two hours old or older. It also deletes sensor readings from `leavingTimes` that are half an hour old or older. With the remaining readings, it calculates and returns the average elapsed time for cars that have times recorded by both sensors.

     There are two corner cases to mention. **COMP 200** students do *not* need to deal with these, while **COMP 130** students do.

     – A car could cross the first sensor before midnight and the sensor sensor after midnight. The resulting transit time shouldn't be negative.

     – If there have been no cars travelling between both sensors recently, calculating the average elapsed time is meaningless. Instead, the special value `None` should be returned.

   In order to test your answer, you should create multiple series of function calls, the results of which you have pre-computed. To get you started, we have provided a sample series and its expected results, along with explanatory comments.

```
sensor1("car1",5.5)
sensor1("car2",6)
sensor1("car3",6)
sensor1("car4",6)
sensor2("car1",6)
print update(6.1)   # No data is old enough to delete.
                    # Print .5, the transit time of car1.
sensor2("car5",6.3) # Entered highway after sensor 1.
sensor2("car3",6.3)
sensor2("car2",6.4) # Cars don't have to enter and leave in same order.
```

```
print update(6.4)   # No data is old enough to delete.
                    # Print .4, the average transit time of cars 1,2,3.
print update(6.5)   # Throw away sensor 2 reading for car 1.
                    # Print .35, the average transit time of cars 2,3.
print update(6.8)   # Throw away sensor 2 readings for cars 3,5.
                    # Print .4, the transit time of car 2.
sensor1("car6",7.4)
sensor2("car6",7.7)
print update(8)     # Throw away sensor 1 readings for cars 1,2,3,4.
                    # Throw away sensor 2 reading for car 5.
                    # Print .3, the transit time of car 6.
print update(9)     # Throw away sensor 2 reading for car 6.
                    # Print None.
```

As an aside, this simulates one particular method for measuring traffic times. Another approach is to use RADAR or other kinds of sensors that can measure speed at a particular location. The following paper provides a good summary of current techniques of traffic detection and measurement:

http://ftp.jrc.es/EURdoc/JRC47967.TN.pdf

Edit Distance – COMP 130 (50 points)

5. (50 points)

   Write a function called `edit_dist(s1, s2)` that computes the Levenshtein edit distance from `s1` to `s2` using the bottom-up approach detailed in the class lecture web page:

   http://www.clear.rice.edu/comp130/12spring/editdist/

   The function `edit_dist` should return both the edit distance and the edit distance matrix in that order. The `s1` substrings should be represented by the rows of the matrix and the `s2` substrings by its columns.

   Note that this function is slightly different than `edit_dist_matrix function` provided at the end of that web page because it returns the edit distance value as well.

   Your code should pass all the tests in OwlTest's Assignment 3 at

   http://www.clear.rice.edu/owltest

Feedback – COMP 200 & COMP 130 (0 points)

1. Roughly how many hours did you spend on the two sets of finger exercises?
2. On a scale of 1 (very easy) to 5 (very difficult), how difficult were the finger exercises?
3. Roughly how many hours did you spend on this homework?
4. On a scale of 1 (very easy) to 5 (very difficult), how difficult was this homework?
5. Which material did you find most challenging?
6. Did you feel that the class material adequately prepared you for the homework?