CSCI 1100 — Computer Science 1 Homework 7
Dictionaries

Overview

This homework is worth **100 points** toward your overall homework grade, and is due Thursday, November 14, 2019 at 11:59:59 pm. It has two parts, each worth 50 points. Please download `hw7_files.zip`. and unzip it into the directory for your HW7. You will find multiple data files to be used in both parts.

The goal of this assignment is to work with dictionaries. In part 1, you will do some simple file processing. Read the guidelines very carefully there. In part 2, we have done all the file work for you so you should be able to get the data loaded in just a few lines. For both parts, you will spend most of your time manipulating dictionaries given to you in the various files.

Please remember to name your files `hw7_part1.py` and `hw7_part2.py`.

As always, make sure you follow the program structure guidelines. You will be graded on program correctness as well as good program structure.

Remember as well that we will be continuing to test homeworks for similarity. So, follow our guidelines for the acceptable levels of collaboration. You can download the guidelines from the Course Resources section of Submitty if you need a refresher. Note that this includes using someone else's code from a previous semester. Make sure the code you submit is truly your own.

Autocorrect, now improved a bit more...

As promised, this is a simple modification of the HW6 version of autocorrect. You will make a few changes to make it more realistic. We will describe the whole homework but point out the differences in bold. Feel free to start from your HW6 code. Do not start from someone else's HW6 code.

To solve this problem, your program will read the name of **three** files:

- the first contains a dictionary of words (**Note: the format of the file is changed**)
- the second contains a list of words to autocorrect (as before)
- the third (**a new file**) contains potential letter substitutions (described below).

The input word file has a single word per line as before, but the dictionary file has two entries per line, the first entry on the line is a single valid word in the English language and the second entry is a float representing the frequency of the word in the lexicon. The two values are separated by a comma. **The inclusion of frequency is a slight change from the dictionary used in the previous assignment.**

Read the English dictionary into a Python dictionary, using words as keys and frequency as values. You will use the frequency for deciding the most likely correction.

The keyboard file has a line for each letter. The first entry on the line is the letter to be replaced and the remaining letters are possible substitutions for that letter. All the letters on the line are separated by spaces, These substitutions are calculated based on adjacency on the keyboard, so if you look down at your keyboard, you will see that the "a" key is surrounded by "q", "w", "s", and "z". Other substitutions were calculated similarly, so

```
b v f g h n
```

means that a possible replacement for `b` is any one of `v f g h n`. Read this keyboard into a dictionary: the first letter is the key (eg. `b`) and the remaining letters are the value, stored as a list.

Your program will then go through every single word in the input file, autocorrect each word and print the correction. To correct a single word, you will consider the following:

FOUND If the word is in the dictionary, it is correct. There is no need for a change. Print it as found, and go on to the next word. (No change: you are now checking the word is in the key for your dictionary: use `in dictionary` to test please. Do not use a loop.)

DROP If the word is not found, consider all possible ways to drop a single letter from the word. **Store any of the words that are in your English dictionary in some container (list/set/dictionary). Note that you do not stop if a match is found here. Read below for ranking all matches.**

INSERT If the word is not found, consider all possible ways to insert a single letter in the word. **Store any of the words that are in your English dictionary in some container (list/set/dictionary). Note that you do not stop if a match is found here. Read below for ranking all matches.**

SWAP Consider all possible ways to swap two consecutive letters from the word. **Store any of the words that are in your English dictionary in some container (list/set/dictionary). Note that you do not stop if a match is found here. Read below for ranking all matches.**

REPLACE Next consider all possible ways to change a single letter in the word with any other letter **from the possible replacements in the keyboard file only. Store any of the words that are in your English dictionary in some container (list/set/dictionary).** This means that now, you will have fewer potential replacements but more likely ones. (**Note that you do not stop if a match is found here. Read below for ranking all matches.**)

For example, for the keyboard file we have given you, possible replacements for `b` are `v f g h n`. Hence, if you are replacing `b` in `abar`, you should consider: `avar, afar, agar, ahar, anar`.

If there are multiple potential matches, sort them by their potential frequency from the English dictionary and return the top 3 values that are in most frequent usage as the most likely corrections in order. If there are fewer than 3 potential matches, print all of them in order. You do not have to print how this match was found (drop, swap or replace). In the unlikely event that two words are equally as likely as frequent, you should pick the one that comes last in lexicographical order, Refer to the section on "General hint on sorting" for how to do this easily.

NOT FOUND If there are no potential matches using any of the above corrections, print NOT FOUND.

When printing words and matches, format them to to 15 characters.

Here is a potential output of your program for the English dictionary we have given you (note that as usual, we will use a more extensive dictionary on Submitty. Your results will vary during submission):

```
Dictionary file => words_10percent.txt
words_10percent.txt
Input file => input_words.txt
input_words.txt
Keyboard file => keyboard.txt
keyboard.txt
barely          -> barely          :FOUND
carats          -> carats          :FOUND
confabs         -> confabs         :FOUND
corepulent      -> corpulent       :MATCH 1
costumye        -> costumey        :MATCH 1
pamek           -> pamek           :NO MATCH
darjed          -> darned          :MATCH 1
darjed          -> dared           :MATCH 2
darjed          -> darked          :MATCH 3
doitd           -> doit            :MATCH 1
doitd           -> doits           :MATCH 2
dovrtailing     -> dovetailing     :MATCH 1
flavoonls       -> flavonols       :MATCH 1
outstpend       -> outspend        :MATCH 1
hut             -> shut            :MATCH 1
hut             -> jut             :MATCH 2
pashas          -> pashas          :FOUND
poolry          -> poorly          :MATCH 1
quixmasters     -> quizmasters     :MATCH 1
relocatiing     -> relocating      :MATCH 1
sorceresd       -> sorceress       :MATCH 1
turbulences     -> turbulences     :FOUND
sut             -> shut            :MATCH 1
inteters        -> integers        :MATCH 1
marteatment     -> marteatment     :NO MATCH
agew            -> ages            :MATCH 1
agew            -> agee            :MATCH 2
agew            -> anew            :MATCH 3
shu             -> shut            :MATCH 1
shu             -> shy             :MATCH 2
shu             -> shh             :MATCH 3
```

When you are sure your homework works properly, submit it to Submitty. Your program must be **named** `hw7_part1.py` to work correctly.

To finish up, consider how we made different design decisions that resulted in different outputs. What is the best design for this problem? What other changes should be considered? The more changes you consider, the further you will be from the word the author intended. Should certain operations be first? We used the words that are common in English, but we should consider words that are common to the person typing them. This is of course how it works on your mobile devices. Hopefully, this homework showed you how the every day tools you use are based on simple ideas put together in different ways.

Well rated and not so well rated movies ...

In this section, we are providing you with two data files `movies.json` and `ratings.json` in JSON data format. The first data file is movie information directly from IMDB, including rating for some movies but not all. The second file is ratings from Twitter. **Be careful:** Not all movies in `movies.json` have a rating in `ratings.json`, and not all movies in `ratings.json` has relevant info in `movies.json`.

The data can be read in its entirety with the following five lines of code:

```python
import json

if __name__ == "__main__":
    movies = json.loads(open("movies.json").read())
    ratings = json.loads(open("ratings.json").read())
```

Both files store data in a dictionary. The first dictionary has movie ids as keys and a second dictionary containing an attribute list for the movie as a value. For example:

```python
print (movies['3520029'])
```

(movie with id `'3520029'`) produces the output:

```
  {'genre': ['Sci-Fi', 'Action', 'Adventure'], 'movie_year': 2010,
    'name': 'TRON: Legacy', 'rating': 6.8, 'numvotes': 254865}
```

This is same as saying:

```python
movies = dict()
movies['3520029'] = {'genre': ['Sci-Fi', 'Action', 'Adventure'],
                     'movie_year': 2010, 'name': 'TRON: Legacy',
                     'rating': 6.8, 'numvotes': 254865}
```

If we wanted to get the individual information for each movie, we can use the following commands:

```python
print (movies['3520029']['genre'])
print (movies['3520029']['movie_year'])
print (movies['3520029']['rating'])
print (movies['3520029']['numvotes'])
```

which would provide the output:

```
['Sci-Fi', 'Action', 'Adventure']
2010
6.8
254865
```

The second dictionary again has movie ids as keys, and a list of ratings as values. For example,

```python
print (ratings['3520029'])
```

(movie with id `'3520029'`) produces the output:

```
[6, 7, 7, 7, 8]
```

So, this movie had 5 ratings with the above values.

Now, on to the homework.

Problem specification

In this homework, assume you are given these two files called `movies.json` and `ratings.json`. Read the data in from these files.

Ask the user for a year range: `min year` and `max year`, and two weights: `w1` and `w2`.

Find all movies in `movies` made between min and max years (inclusive of both min and max years).

For each movie, compute the combined rating for the movie as follows:

`(w1* imdb_rating + w2 * average_twitter_rating)/(w1+w2)`

where the `imdb_rating` comes from `movies` and `average_twitter_rating` is the average rating from `ratings`.

If a movie is not rated in Twitter, or if the Twitter rating has fewer than 3 entries, skip the movie.

Now, repeatedly ask the user for a genre of movie and return the best and worst movies in that genre based on the years given and the rating you calculated. Repeat until the user enters stop.

A sample output of your program is given below (the second lines for each movie has a tab at the start of the line, and the rating is given in `{:.2f}` format.):

```
Min year => 2000
2000
Max year => 2016
2016
Weight for IMDB => 0.7
0.7
Weight for Twitter => 0.3
0.3

What genre do you want to see? sci-fi
sci-fi

Best:
	Released in 2014, Interstellar has a rating of 8.70

Worst:
	Released in 2015, Fantastic Four has a rating of 4.72

What genre do you want to see? drame
drame

No Drame movie found in 2000 through 2016

What genre do you want to see? drama
drama

Best:
        Released in 2008, The Dark Knight has a rating of 9.20

Worst:
        Released in 2012, Spring Breakers has a rating of 4.91

What genre do you want to see? stop
stop
```

We have of course given a smaller subset to you. We will be using a different set for testing on Submitty as usual.

When you are sure your homework works properly, submit it to Submitty. Your program must be **named** `hw7_part2.py` to work correctly.

General hint on sorting

It is possible that two movies have the same rating. Consider the following code:

```
>>> example = [(1, "b"), (1, "a"), (2, "b"), (2, "a")]
>>> sorted(example)
[(1, 'a'), (1, 'b'), (2, 'a'), (2, 'b')]
>>> sorted(example, reverse=True)
[(2, 'b'), (2, 'a'), (1, 'b'), (1, 'a')]
```

Note that the sort puts tuples in order based on the index 0 value first, but in the case of ties, the tie is broken by the index 1 tuple. (If there were a tie in both the index 0 and the index 1 tuple, the sort would continue with the index 2 tuple if available and so on.) The same relationship holds when sorting lists of lists. To determine the worst and best movies, the example code used a sort with the rating in the index 0 spot and with the name of the movie in the index 1 position. Keep this in mind when you are determining the worst and best movies.