CSCI 1100 — Computer Science 1 Homework 7
Dictionaries

Overview

This homework is worth **100 points** and it will be due Thursday, November 14, 2024 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.

Honor Statement

There have been a number of incidents of academic dishonesty on homework assignments and this must change. Cases are easily flagged using automated tools, and verified by the instructors. This results in substantial grade penalties, poor learning, frustration, and a waste of precious time for everyone concerned. In order to mitigate this, the following is a restatement of the course integrity policy in the form of a pledge. **By submitting your homework solution files for grading on Submitty, you acknowledge that you understand and have abided by this pledge:**

- **I have not shown my code to anyone in this class, especially not for the purposes of guiding their own work.**
- **I have not copied, with or without modification, the code of another student in this class or who took the class in a previous semester.**
- **I have not used a solution found or purchased on the internet for this assignment.**
- **The work I am submitting is my own and I have written it myself.**
- **I understand that if I am found to have broken this pledge that I will receive a 0 on the assignment and an addiitional 10 point overall grade penalty.**

You will be asked to agree to each of these individual statements before you can submit your solutions to this homework.

Please understand that if you are one of the vast majority of the students who follow the rules and only work with other students to understand problem descriptions, Python constructs, and solution approaches you will not have any trouble whatsoever.

Part 1: Autocorrect

We have all used auto-correct to fix our various typos and mistakes as we write, but have you ever wondered how it works? Here is a small version of autocorrect that looks for a few common typographical errors.

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

- the first contains a list of valid words and their frequencies,
- the second contains a list of words to autocorrect, and
- the third contains potential letter substitutions (described below).

The input word 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.

Read this 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 when there are multiple possibilities

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 file into a dictionary: the first letter is the key (e.g., `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.

Otherwise consider **all** of the remaining possibilities.

DROP If the word is not found, consider all possible ways to drop a single letter from the word. Store any valid words (words that are in your English dictionary) in some container (list/set/-dictionary). These will be candidate corrections.

INSERT If the word is not found, consider all possible ways to insert a single letter in the word. Store any valid words in some container (list/set/dictionary). These will be candidate corrections.

SWAP Consider all possible ways to swap two consecutive letters from the word. Store any valid words in some container (list/set/dictionary). These will be candidate corrections.

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.** Store any valid words in some container (list/set/dictionary). These will be candidate corrections.

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`.

After going through all of the above, **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 three or fewer potential matches, print all of them in order. In the unlikely event that two words are equally likely based on frequency, you should pick the one that comes **last** in lexicographical order. See the note below.

If there are no potential matches using any of the above corrections, print NOT FOUND. Otherwise, print the word (15 spaces), the number of matches, and at most three matches, all on one line. An example output of your program for the English dictionary we have given you is contained in `part1_output_01.txt`. Note that, we will use a more extensive dictionary on Submitty, so your results may be different on Submitty than they are on your laptop.

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

Notes:

1. Do NOT write a for loop to search to see if a string (word or letter) is in a dictionary! This will be very slow and may cause Submitty to terminate your program (and you to lose substantial points). Instead, you must use the **in** operator.

2. It is possible, but unlikely, that a candidate replacement word is generated more than once. We recommend that you gather all possible candidate replacements into a set before looking them up in the dictionary.

3. Ordering the potential matches by frequency can be handled easily. For each potential match, create a tuple with the frequencey first, followed by the word. Add this to a list and then sort the list in reverse order. For example, if the list is `v`, then you just need the line of code

```
v.sort(reverse=True)
```

Part 2: 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 ratings for some movies but not all. The second file contains ratings from Twitter. **Be careful:** Not all movies in `movies.json` have a rating in `ratings.json`, and not all movies in `ratings.json` have relevant info in `movies.json`.

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

```
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:

```
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:

```
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:

```
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,

```
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.

An example of the program run (how it will look when you run it using Spyder) is provided in file hw7_part2_output_01.txt (the second line for each movie has 8 spaces at the start of the line, and the rating is given in `{:.2f}` format).

The movies we are giving you for testing are a subset of the movies we will use during testing on Submitty, so do not be surprised if there are differences when you submit.

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.