CSCI 1100 — Computer Science 1 Homework 6
Sets and Files

Overview

This homework is worth **100 points** toward your overall homework grade, and is due Thursday, November 7, 2019 at 11:59:59 pm. It has two parts, each worth 50 points. Please download `hw06_files.zip`. and unzip it into the directory for your HW 6. You will find multiple data files to be used in both parts.

The goal of this assignment is to work with sets and files. Most of the file processing is pretty simple. The actual work lies in manipulating the data from files using sets. You must use sets in both parts; both to get points but also to make sure your program runs in a reasonable amount of time. An incorrect implementation that does not use sets, or one that inadvertently converts from sets to lists will consume unnecessary amounts of processing power on Submitty and may not complete, especially for part 1.

Please remember to name your files **hw6_part1.py** and **hw6_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 homework assignments for similarity, so follow our guidelines for the acceptable levels of collaboration. You can download the guidelines from the Course Materials section on Submitty if you need a refresher. Note that guidelines also forbid using someone else's code from a previous semester. Make sure the code you submit is truly your own.

Autocorrect fail...

How does autocorrect work? We use it almost every day! The idea behind it is very simple. If you type a word that is not in my dictionary, I go through all possible ways you could have misspelled a word (within reason) and check whether any of the corrections of misspellings is in my dictionary, if so I return it as my prediction.

Fair warning: we will solve a simplified version of autocorrect in this part. You must use sets. We will revisit the same problem in HW 7 with a more complex solution using dictionaries. You will be able to reuse most of what you write here at that time, so spend some time to structure your code well, and use functions to make it more modular. Think about the future you of two weeks from now trying to read and modify your code, and be nice to that person!

To solve this problem, your program will read the name of two files: the first containing a dictionary of words and the second containing a list of words to autocorrect. Both files have a single word per line.

Your program will read the words from the dictionary into a set. All operations that check for membership of words in the dictionary or for finding the common words within a dictionary must be done with sets.

Your program will then go through every single word in the input file and autocorrect each word. 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.

DROP If the word is not found, consider all possible ways to drop a single letter from the word. If any of them are in the dictionary, then print the word as corrected with drop, and stop. For example, `quinecunx` can be changed to `quincunx` by dropping `e`.

SWAP If the word is not yet corrected, consider all possible ways to swap two consecutive letters from the word. If any of one of these in the dictionary, then print the word as corrected with swap, and stop. For example, `serednipity` can be transformed to `serendipity` by swapping the letters `d` and `n`.

REPLACE If the word is not yet corrected, consider all possible ways to change a single letter in the word with any other letter from the alphabet. You can hardcode a list of all letters in the alphabet for this part:

```
letters = [ a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k',
            'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
            'w', 'x', 'y', 'z']
```

If any of these are in the dictionary, then print it as corrected with replace, and stop. For example, `sockpolager` can be changed to `sockdolager` by replacing `p` with `d`.

NO MATCH If the word is not corrected by any of the above steps, print NO MATCH.

If there are multiple matches in any step, return the match that is smallest lexicographically. This is not super meaningful, but it will do for now. Assume words only contain English letters, no hyphens, quotations, or other non-alphabetical symbols.

Implement the potential ways to do autocorrection in the order given to you: Match, Drop, Swap, and Replace. You will get plenty of partial credit for all of your possible matches. When printing words and matches, format them to 15 characters.

An example of the program run (how it will look when you run it using Spyder IDE) is provided in file hw6_part1_output_01.txt. (In order to access this file, you will need to download file `hw06_files.zip` from the `Course Materials` section of Submitty and unzip it into your directory for HW 6.) Note that this example output uses a specific dictionary file. We will use a more extensive dictionary on Submitty, so your results will vary during submission.

When you are sure your homework works properly, submit it to Submitty. Your program **must be named hw6_part1.py** to work correctly.

To finish up, you can see problems with this approach. Some later matches may have been better than earlier ones for some words, but we considered them in the strict order. We also did not consider some potential misspellings like adding a letter or phonetic misspellings of werdz! We still would like to limit the potential changes to the most likely ones, like keeping in mind what keyboard people are using or which words are more common within the language. We will address some of these in the next homework by adding to this solution.

Fantastic beasts in the wizarding world of CS1...

In this part, you will use a file containing various beasts that were featured in different titles in the Harry Potter series of books and movies. We give you a simple data file called `titles.txt` to test your program, but we will be working with a more comprehensive one on Submitty.

Each line of the file contains a title (movie or book), followed by all the beasts featured in this title. For example, the following line:

`Harry Potter and the Goblet of Fire|Hippocampus|Merpeople|Niffler`

mentions that the title `Harry Potter and the Goblet of Fire` featured three beasts: `Hippocampus`, `Merpeople`, and `Niffler`. You can assume that the data is valid and that splitting on the `'|'` string will correctly parse the lines in `titles.txt`.

Your program must do the following:

- Read the name of a title from the user. The user will enter any part of the title (in any case, upper or lower). Find the first title in the list that contains the input string. For example, the user may enter `FIRE`, which will match the above line.

  If no match is found, print the message.

  If a match is found, then do the following:

  - Print all the beasts that were featured in this title in lexicographical order.
  - Print all the other titles that have at least one beast in common with this title in lexicographical order.
  - Print all the beasts that were featured only in this title (i.e., no other title has these beasts), again in lexicographical order.

    Consider which set operation will help with the last two operations ...

- Once you have finished processing the input (match or no match), ask for another input, until the user inputs `stop`.

Your output will likely contain long lists of values. It must be formatted nicely! If your output is correct but is not formatted properly, you will lose some points but not all. However, Submitty may not catch this and you may only get points back when your TA reads your homework. In either case, your formatting must be identical to that expected in Submitty to get full credit.

Each list you print should be separated with a comma and a space. You will find that using `print()` to achieve this is difficult. It is easier to construct a string with the information you want, for example: `line = val1 + ", " + val2`, and then print the line.

If you have a very long line, you can use the `textwrap` module to break it into a list of multiple lines. Here is an example use of this module.

```
>>> import textwrap
>>> par = "this is a very long line with many unnecessary words using it to show how
    textwrap works"
>>> lines = textwrap.wrap(par)
>>> lines
['this is a very long line with many unnecessary words using it to show', 'how textwrap
    works']
>>> for line in lines:
...     print(line)
...
this is a very long line with many unnecessary words using it to show
to how textwrap works
```

An example of the program run (how it will look when you run it using Spyder IDE) is provided in file hw6_part2_output_01.txt.

When you are sure your homework works properly, **name it correctly as hw6_part2.py** and submit it to Submitty.