Project 4 CPE 202 Fall 2018

Concordance (An application of hash tables)

This assignment has several parts: implementing a hash table similar to those described in the text (with some additional functionality) and writing an application that builds a concordance. A Webster's dictionary definition of concordance is: "an alphabetical list of the main words in a work." In addition to the main words, your program will keep track of all the line numbers where these main words occur.

Word and Line Concordance Application

The goal of this assignment is to process a textual data file to generate a word concordance with line numbers for each main word. A dictionary ADT is perfect to store the word concordance with the word being the dictionary key and a list of its line numbers being the associated value for the key. Since the concordance should only keep track of the "main" words, there will be another file containing words to ignore, namely a **stop-words file** (stop_words.txt). The **stop-words file** will contain a list of stop words (e.g., "a", "the", etc.) -- these words will not be included in the concordance even if they do appear in the data file. You should also not include strings that represent numbers. e.g. "24" or "2.4" should not appear.

The following is an example and the output file.

Sample `stop_words.txt` file

```
a
about
be
by
can
do
i
in
is
it
of
on
the
this
to
was
```

Sample `hw5data.txt` file

```
This is a sample data (text) file to
be processed by your word-concordance program.

The real data file is much bigger.
```

Notes:

1) Words are defined to be sequences of letters delimited by any non-letter. (e.g., white space, punctuation, parentheses, dashes, double quotes, etc.)
2) There is to be no distinction made between upper and lower case letters. (e.g., "CAT" is the same word as "cat")
3) Blank lines are to be counted in the line numbering. (e.g., line 3 above is blank)

Sample output file

```
bigger: 4
concordance: 2
data: 1 4
file: 1 4
much: 4
processed: 2
program: 2
real: 4
sample: 1
text: 1
word: 2
your: 2
```

The general algorithm for the word-concordance program is:

1) Read the `stop_words.txt` file into **your implementation of a hash table** containing the stop words. For the initial table size, start with default of 191 and let the table grow as described below, if necessary. Note: You should use the same hash table implementation for the stop-words and the concordance. In the case of the stop-words, you just won't use the line number information (can either store the actual line number from the file, or just use a default value). (WARNING: Make sure you do not include the newline character ('\n') as part of the word when adding the stop words.)

2) The word-concordance dictionary will be in a separate hash table from the stop words hash table. Process the `input` file one line at a time to build the word-concordance dictionary. This hash table should only contain the **non-stop words** as the keys (use the stop words hash table to "filter out" the stop words). Associated with each key is its **value** where the **value** consists of a list containing the line numbers where the key appears. DO NOT INCLUDE DUPLICATE LINE NUMBERS.

3) Generate a text file containing the concordance words printed out **in alphabetical order** along with their line numbers. One word per line (followed by a colon), and spaces separating items on each line:

```
data: 1 4
```

Note there is no space after the last line number - make sure to match the sample output files.

It is strongly suggested that the logic for reading words and assigning line numbers to them be developed and tested separately from other aspects of the program. This could be accomplished by reading a sample file and printing out the words recognized with their corresponding line numbers without any other word processing.

Collision resolution:

- Your implementation should use Open Addressing using quadratic probing for collision resolution.
- Note that you will not have to incorporate deleting items from your hash table

The hash function should take a string containing one or more characters and return an integer. Use Horner's rule to compute the hash efficiently:

h(str) = $\sum_{i=0}^{n-1} ord(str[i]) * 31^{n-1-i}$ where n = the **minimum** of **len(str) and 8**

Also, your hash table size should have the capability to grow if the input file is large. After insertion of an item, if the load factor exceeds 0.5, you should grow the hash table size.
Start with a default hash table size of 191, then if increases are necessary, use:
"new table size" = 2 *"old table size" + 1 (use this "new table size" even if it is no longer a prime)

Provided Data Files

- the stop words in the file `stop_words.txt`
- six sample data files that can be used for preliminary testing of your programs:
  - `file1.txt, file1_sol.txt` - contains no punctuation to be removed
  - `file2.txt, file2_sol.txt` - contains punctuation to be removed
  - `declaration.txt, declaration_sol.txt` – larger file for test

Your code for creating the concordance will be contained in two files: hash_quad.py and concordance.py

The hash_quad.py file will contain the HashTable class and the methods described below for that class.
Each entry in the hash table will consist of a string (a word) and list of line numbers in which the word appears. How you implement an "entry" is your design choice. You could use a Python tuple for each entry with the first element as the key (word) and the second element as the value (list of line numbers), for example:
`("cat", [1, 3, 8])`
Another possibility would be to define a class for an entry that has key and value attributes.

The following starter files (containing the methods that you must implement) are available on GitHub. **Do not change the names of the classes, methods, or attributes specified in the starter files.**

- hash_quad.py
- concordance.py

SUBMISSION

Six files:

- **hash_quad.py**: containing class HashTable with all of the specified methods, using quadratic probing for collision resolution
- **hash_quad_tests.py** – tests for the HashTable methods specified by the assignment
    - This file should contain tests only for the functions specified above and should only test functionality required by the specification. These tests must run properly on a valid instructor solution and will be tested to see if they can catch bugs in incorrect solutions.
- **hash_quad_helper_tests.py** – tests for the HashTable methods specified by the assignment
    - This file should contain all tests for your solution, including tests for any helper functions that you may have used. These tests will be used to verify that you have 100% coverage of your solution. Your solution must pass these tests.
- **concordance.py**: containing class Concordance with all of the specified methods
- **concordance_tests.py** – tests for the Concordance methods specified by the assignment
    - This file should contain tests only for the functions specified above and should only test functionality required by the specification. These tests must run properly on a valid instructor solution and will be tested to see if they can catch bugs in incorrect solutions.
- **concordance_helper_tests.py** – tests for the Concordance methods specified by the assignment
    - This file should contain all tests for your solution, including tests for any helper functions that you may have used. These tests will be used to verify that you have 100% coverage of your solution. Your solution must pass these tests.

Helpful resources:

- A "How to on sorting in Python": https://docs.python.org/3/howto/sorting.html . You may use the built-in sorting routines in Python so you may find this reference helpful.
- Python has some built in capability to eliminate punctuation that you may find helpful. See string.constants https://docs.python.org/3.1/library/string.html (apostrophes should be removed, hyphens should be replaced by a space – other punctuation can be removed or replaced by a space)