Assignment 4 CSC 202 Fall 2017

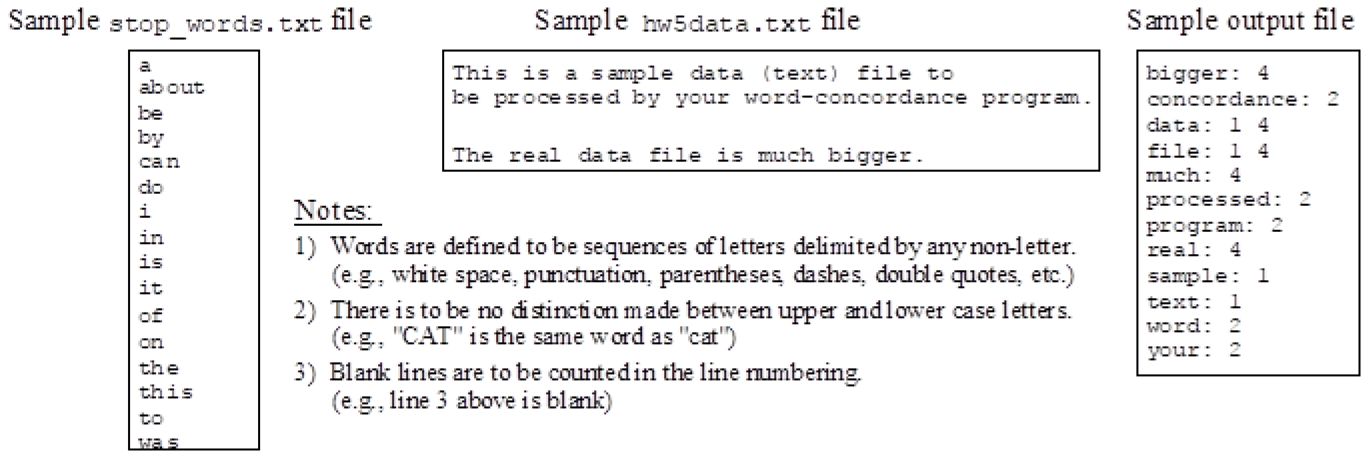
1 Concordance (an application of hash tables)

This assignment has several parts: implementing different versions of the hash table from 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 a second 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 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.
```

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
```

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)

The general algorithm for the word-concordance program is:

1) Read the `stop_words.txt` file into **your implementation of a hashtable** containing only stop words. For the initial table size, you could choose the next prime number after two times the number of stop words, or just start with default of 251 and let the table grow as described below. (WARNING: Make sure you do not include the newline character ('\n') as part of the word when adding the stop words.)

2) Process the `input` file one line at a time to build the **word-concordance dictionary**. This dictionary should contain the **non-stop words** as the keys. 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. See the sample output files.

(Note: 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.)

DICTIONARY ADT COMPARISON

Implement 2 dictionary ADT implementations:

- Open Addressing using linear probing
- Open Addressing using quadratic probing

Your basic hash function should take a string containing 1 or more characters and return an integer. Use Horner's rule to compute the hash efficiently. See below.
Also your hashtable size should have the capability to grow if the input file is large. You should start with a default hash table size of 251, then 503, then if further increases are necessary, use "new table size" = 2 *"old table size" + 1 , and use this "new table size" even if it is no longer a prime.

DATA FILES –

- the stop words in the file `stop_words.txt`
- four sample data files that can be used for preliminary testing of your programs:
    - input1.txt, concord1.txt - contains no punctuation to be removed
    - input2.txt, concord2.txt - contains punctuation to be removed

The hash table classes you implement **should each contain** the following methods with exact signatures below. These methods may be called in grading your program

- **def __init__ (self, size)**: creates and initializes the hash table size to size
- **def read_stop (self, filename):** read words from a stop words file and insert them into hash table
- **def read_file (self, filename, stop_table):** read words from input file and insert them into hash table, after processing for punctuation, numbers and filtering out words that are in the stop_table
- **def get_tablesize(self):** returns the size of the hash table
- **def save_concordance(self, outputfilename):** see sample output files for format
- **def get_load_fact(self):** returns the load factor of the table
- **def myhash(self, key, table_size)** and return an integer from 0 to the (size of the hash table) – 1. Compute the hash value by using Horner's rule: h_value(str) = $\sum_0^{n-1} ord(str[i]) * 31^{n-1-i}$ where n = the **minimum** of **len(key) and 8**.

SUBMISSION

Two files: (these two files will contain the same functions, but use different probing as described above)

- hash_lin_table.py containing -- class HashTableLinPr, using linear probing
- hash_quad_table .py containing -- class HashTableQuadPr, using quadratic probing

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 build 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)
- The following is a simple approach to test if a string contains a number:

```
def is_number(s):
    try:
        float(s)
        return True
    except ValueError:
        return False
```