CPSC2108 Data Structures, Spring 2014

ASSIGNMENT 5 – DUE BEFORE MIDNIGHT, WEDNESDAY, APRIL 23

Anagram Queries using Hashing[1]

The purpose of this assignment is to understand and apply hashing as an efficient search technique. Hashing is covered in sections 7.3 and 7.4 of the text book.

Write a program to which prints out all anagrams of a specified string. Two strings are anagrams of one another if by rearranging letters of one of the strings you can obtain another. For example, the string "toxic" is an anagram of "ioxct". For the purpose of this assignment, we are only interested in those anagrams of a given string which appear in the dictionary. The dictionary you should use is the file available here.

Algorithm and Implementation

Since we will be performing multiple anagram queries, our first step is to load all of the (25,000) words in the dictionary into an appropriate data structure. A primary requirement is that one must be able to efficiently search this data structure to look for anagrams of a given word. A clever trick that we will use to facilitate this is to first sort the letters of every word we insert into our data structure to produce a key for each word. You may use any sort you wish -- including one from the Java API (A quick tutorial on this is available at: http://www.javaprogrammingforums.com/java-se-api-tutorials/182-how-sort-array-using-java-util-arrays-class.html). For example, the key for the string "toxic" is "ciotx", similarly the key for both "star" and "rats" is "arst". We will then use a hash table to store pairs of strings, where the pair consists of the original word and its key. You might want to write a little class to represent a hash table entry consisting of such a pair.

When performing insertions into the hash table, we will first compute the key of the word to insert and then compute the hash of the key to compute the correct bucket (location in the hash table). This approach guarantees that all words which are anagrams of one another are stored in the same bucket of the hash table. Similarly, when we are searching for anagrams, we will first compute the key of the word we are searching for, then hash the key, then search that bucket for anagram matches. Use the Java `String.hashcode()` method for hashing strings. If it produces collisions, handle the collisions using either open addressing or chaining.

Details

[1] Adapted from http://penguin.ewu.edu/cscd300/Summer_11/Assignments/prog4.html

The hash table code which you provide only needs to have the minimum functionality needed to solve this assignment. You may fix a size for your hash table for efficient searching. I recommend that the final hash table you submit contain at least 25,000 buckets. (For debugging your code, I suggest you work with a much smaller practice dictionary, perhaps 10 words, and a much smaller hash table, perhaps 8-10 buckets (depending on whether or not there are any anagrams in the dictionary). To help reduce collisions, make sure your table size is a prime number. Remember it is ok to sacrifice space for speed. That said, your table should not be bigger than 200,000.
You may disregard any words in the dictionary which contain any punctuation characters. Also, you should convert any uppercase characters to lowercase (thus you are only representing words that contain all lower case characters).
Your program should read anagram queries from a file. Each query in the file will be on its own line and will simply consist of a string. The output file should contain the original string, then the number of matching anagrams, followed by those anagrams. An example input file and the resulting output file are attached to this assignment dropbox. Your output file should match this format exactly, except that the matching anagrams you output may be ordered differently.
Do not count a word as an anagram of itself.

What to submit:

- A single zipped folder named in the format: *<your last name_initial_Assignment 4>.zip*, containing the following:
  - ✓ The .jar file (including all source code files and your project file)
  - ✓ the words file (the dictionary of words)
  - ✓ the input file you used for testing
  - ✓ the output file your program produced

  It is your responsibility to make sure your jar file can be extracted and the code runs on a different computer.
  Submit your zip file using CougarView Assignment 5 drop box before midnight, Wednesday, April 23. You must acknlowledge the source of any code not written by you. You must also mention in the comments box in CougarView any problems encountered by you.

Grading Rubric:

1	Good documentation (follow style given in textbook program listings) • Header documentation • Class and method documentation using the Javadoc style followed in the text book	10 points
2	Good programming style • Good use of white space • Meaningful variable names • Use of indentation	10 points
3	Robust • Will not crash with unexpected input • Will not crash if a data file is not found	10 points
4	Program • Fulfills all requirements • Solves the problem • Good program logic • Good use of data structures • Efficient algorithm design	60 points
5	Sample runs • Input and output files of sample runs are included to adequately demonstrate the program	5 points
6	Instructions followed	5 points

Maximum points awarded for programs that fail to compile: 50 points
Please be aware of the penalties of not following the **academic honesty** guidelines given in page 6 of the course syllabus.

Adapted from http://penguin.ewu.edu/cscd300/Summer_11/Assignments/prog4.html