CMPS 101 Spring 2018
Programing Assignment #3
Fast Find Anagrams

May 6, 2018

Goal

The goal of this assignment is learn how to use Java to design ADTs for the detection of anagrams.

Program Specification

For this assignment you will write a program that given a word list file will find all the anagrams in the list for individual strings of letters entered from the keyboard.

Here is a sample execution:

```
java FindAnagrams wordList.txt
type a string of letters
items
emits
metis
mites
smite
stime
times
Do another (y/n)?
y
type a string of letters
march
charm
Do another (y/n)?
n
```

Getting Started

From a user's point of view the logic of this program is very simple. You get a string of letters from the keyboard and search through a word file looking for words that are anagrams of the input string.

A key operation you will need is to check if a string is an anagram of another string. This of course can be done in a brute force manner by looking at the individual letters in one string one at a time and checking off if they are contained in the other string, etc. This is very time consuming. There is a better way. **For this assignment, in order to receive full credit you will need to find and implement an algorithm that is more efficient than simple brute force comparison against every word in the list.** Consider if you have some way to code which multiset of letters is in a word and you associate such a code with each word then words which are anagrams of each other will have the same code.

To use this idea you need to choose an appropriate coding, create a method to compute the code for a word, and a method to check if the codes for two different words are the same.

A good code should have three characteristics.

1. It should be the same for words that have the same multiset of letters.
2. It should not be the same for words that do not have the same multiset of letters.
3. It should be faster than brute force letter by letter comparison. In particular, the number of comparisons (if-tests) to compare two words should be some small constant (ideally 1), not a function of the number of letters in the word. Notice that in the worst case brute force requires N*(N+1)/2 comparisons for words of length N.

One code which will work is to associate a random number with each of the letters in the alphabet. Then the code for a word is defined to be the sum of the numbers associated with its individual letters. This technique clearly satisfies condition 1 and with high probability satisfies condition 2. You can probably think of other possible codes. We will discuss some others in class.

Correctness Points

- 10 points - compiles without errors and is a serious attempt at a solution
- 10 points - only reads word file from disk once.
- 10 points - for comparisons uses an algorithm faster than brute force, letter by letter comparison for each word in the word list. $T(wordsize) = O(1)$.
- 10 points - correctly identifies anagrams for the input string

- 5 points - does NOT print out the input string as a possible anagram if it happens to be a word (the input string does not need to be a word)
- 5 points - allows for multiple input strings

Requirements

Implement `Anagram` type that provides

- a constructor which uses a String, made up of alphabetic characters either upper or lower case as an input argument.
- a constructor which uses a char array, made up of alphabetic characters either upper or lower case as an input argument.
- a method for printing.
- a method for comparing two Anagram variables that returns true or false.
- a method that returns the word part of an Anagram variable.
- do not allow user to get the code part of an Anagram variable.

Testing Write test programs to verify that your ADT implementation is working correctly. You should have a test for every bullet point in the above list.

Execute test runs of your FindAnagrams program.

Direction and hints

- Use the names `FindAnagrams` for your application program and `Anagram` for your ADT and name your print method `print`. This is mandatory to facilitate testing and grading.
- You can find the file wordList.txt at /afs/cats.ucsc.edu/courses/cmps101-db/wordList.txt note the first word in the file is an integer indicating the total number of words that follow.
- Use good coding style. See `www.soe.ucsc.edu/~sbrandt/105/coding.html` for guidelines.

Submission instructions

- Create directory named `CMPS101S18PA<PROGRAMMING ASSIGNMENT NUMBER>`, e.g. `CMPS101S18PA3` for Programing Assignment 3.
- In the program assignment directory put in the following files

  - `README` - a short file which lists all the files in the directory and describes what they are.
  - `NoteToGrader` - a short note in which you describe your approach.
  - The `<SOURCEFILES>` - the Java files which you wrote in this assignment.
  - The `<TESTFILES>` - the test files which you used to ensure that your program works correctly.

- Go to your directory, and invoke the `script` command. For a tutorial on how to use this command see `http://www-users.cs.umn.edu/~gini/1901-07s/files/script.html`. In your case, you should invoke the following commands:
  - `script pa3submissionfile.txt`
  - `pwd` - this will show us which directory you are in.
  - `ls -l` - this will list the contents of the directory.
  - `cat README`
  - `cat NoteToGrader`
  - `cat <SOURCEFILES>` - this will print the contents of the source files to the screen. We want you to run this command because this is the only way in which we will see the code you wrote. Run this command on every source files you are using, but DO NOT RUN IT ON YOUR BINARY FILES!
  - `<commands to compile/link the program>` - run the commands which build your source code
  - `ls -l` - this will list the contents of the directory again, showing us that there are binaries which resulted from the previous step.
  - `cat <TESTFILES>` - this will print the contents of the test files to the screen. As before, we want you to run this command because this is the only way in which what kind of tests you ran to ensure that the source files are doing what they should be doing. Run this command on every test file you are using, but DO NOT RUN IT ON YOUR BINARY FILES!
  - `<commands to execute required tests>` - run the commands which test your binaries against specific files. Make sure that the results of these tests are printed to screen.
  - `exit` - this will exit the `script` command and produce a plain text called `pa3submissionfile.txt`.

- Take the plain text file which you created in the previous step and paste its contents to a `.pdf` file. If you've never created a `.pdf` file we have a guideline for this on canvas. This `.pdf` file is the only document you will submit on canvas.

- Do not submit source files or binaries!
- Do not run any editing commands between the `script` command and the `exit` command. It will produce a mess in the plain text file.