CS 1501
Algorithm Implementation
Programming Project 1 Part A
(see Part B online soon)

Online: Wednesday, May 17, 2017
Due (BOTH Parts A and B): All assignment materials: 1) All source files of program, 2) All input data files except the dictionary file 3) All output files 4) Well written/formatted paper explaining your anagram algorithm and tree implementation (see Part B for details on the paper) and 5) Assignment Information Sheet. All materials must be zipped into a single .zip file and submitted **via the course submission site** by **11:59 PM on Monday, June 5, 2017**. Note: Do **NOT submit the dictionary file**.
Late Due Date: 11:59PM on Wednesday, June 7, 2017

Note: Do not submit a NetBeans or Eclipse (or any other IDE) project file. If you use one of these IDEs make sure you extract and test your Java files WITHOUT the IDE before submitting.

Background:
Word scrambles (or anagrams) are fun puzzles that challenge our minds and our patience. The general idea of an anagram is that a string of letters is rearranged to produce a valid word or phrase. Originally the game was to unscramble a meaningless string of letters to generate a meaningful word or phrase. More recently (thanks to computers) it is also used to "create" phrases from other words or phrases. For input consider a string of letters which may or may not contain blanks (the blanks will be ignored in the anagram game, but are allowed to make the input easier for the user). Each solution must use all of the letters in the input string (except blanks) but the solution can have an arbitrary number of words in it. The output will be a collection of valid resulting words or phrases, sorted from the phrase with the fewest words to the one with the most. Within groups with the same number of words, the listings should be alphabetical. For example, given the letters:

```
raziber
```

your anagram program should generate the following list of words:

```
bizarre
brazier
raze rib
rib raze
```

(Note: Answers will vary depending on the underlying dictionary used. The answers above are based on the dictionary used in this assignment)
You assignment is to implement this anagram program in the following way:

1) Read a dictionary of words in from a file and form a **MyDictionary** of these words. The interface `DictInterface` and the class `MyDictionary` are **provided for you** on the CS 1501 Web page, and **you must use them** in this assignment. Read over the code and comments carefully so that you understand what they do and how. **The interface DictInterface will also be important for Part B of this assignment.** The file used to initialize the MyDictionary will contain ASCII strings, one word per line. Use the file **dictionary.txt** on the CS1501 Assignments page. To see how DictInterface and MyDictionary work, see the program DictTest.java.
2) Read input strings from an input file (one string per line) and calculate the anagrams of those strings and output them as described above. Your output must go to an output file. The user should be able to enter both the input file name and the output file name on the command line. Your program should read the strings one at a time until it reaches the end of the file. For each input string your program must output all of the anagram strings to the output file in the order specified. **For more specifics on the requirements for generating the anagrams, see the comments below.** See the CS 1501 Assignments page for the input files that you must use and for some sample output files.

For an example of an anagram-finding program, see the following link:
http://www.ssynth.co.uk/~gay/anagram.html
Use this program for reference and to help you in developing your solution, but note a few important things:

- The dictionary used for his program is different from the one you will be using, so in many cases the solutions will not be the same
- The solutions there are not listed in the same order that you are required to follow

Important Notes:

- **Be sure to complete Part B of this Assignment. It will be online soon.**

- I recommend developing the anagram algorithm using a small words file and a known test file. This will allow you to trace the execution by hand if necessary, and will help with the debugging.
- **Here are some important details about the anagram algorithm:** Your anagram algorithm **must be a recursive backtracking algorithm** (see class notes for idea of recursive backtracking algorithms). It must include pruning to reduce the number of possible solutions and must be implemented in a reasonably efficient way (including how your strings are manipulated). To enable you to get more partial credit for the anagram algorithm, you **may** want to consider the algorithm in **two parts**:
  1) Determine the valid anagrams (if they exist) using ALL of the letters in the input in a **single word** (if there were spaces in the input, remove them first) (ex: for the example above the solutions would be `bizarre` and `brazier`). This process can be done with a fairly straightforward recursive / backtracking algorithm that considers various permutations of the input characters. However, you **MUST** prune your execution tree such that impossible permutations are not tried beyond the point where they are known to be impossible. This can be done by testing prefixes in the dictionary. For example, using the input above ("raziber"), if we consider the letters in the order given, we can test in the following fashion:

     r | ra | raz | razi  backtrack at this point because "razi" is not a valid prefix in our word dictionary

     Note that in order to implement this correctly, you need to test if a string is a valid prefix in the dictionary. The searchPrefix() method in DictInterface can do this. **If you complete this part correctly (assuming the rest of your project, including Part B, is also correct) you can receive up to 85 total points for the assignment.**
  2) Once you can find the single-word anagrams using all of the letters, add code that allows you to generate multiple-word anagrams as well (ex: for the above example all of the solutions AFTER `bizarre` and `brazier`). This is tricky since there are different ways of approaching the algorithm and the recursion / backtracking is more complicated. Even storing the solutions requires some thought in this case. Think carefully about how you would approach this before coding it (try it with a pencil and paper). Don't forget to sort the solutions from fewest words to most words (and alphabetically for solutions with the same number of words). **If you complete this part correctly (assuming the rest of your project is also correct) you can receive up to 100 total points for the assignment.**

- If a letter (or letters) appear more than once in the initial string, your algorithm will likely produce some anagrams more than once. Think about why this is the case. However, **duplicates should NOT appear in your solutions**, so be sure to eliminate them before printing your results. Note also that different solutions can have identical words in different orderings – the order of the words is part of the solution. See the example output for a demonstration of this point.

- Your output should match that shown on the CS 1501 Assignments page for the input files shown.

- If you search the Web you will find the anagram program indicated in the link above and others as well. If you search hard enough you can probably find source code to one or more of these programs. I strongly urge you to resist trying to find this code. If you use code found from the Web for this project and you are caught, you will receive a 0 for the project (following the cheating guidelines as stated in the Course Policies).

- Be sure to thoroughly document your code, especially any code whose purpose is not clear / obvious.

- **Be sure to also complete Part B of this project. It will be online soon.**