CS 1501
Algorithm Implementation
Programming Project 1

Online: Friday, May 17, 2019
Due: All assignment materials: 1) All source files of program, 2) All input data files except the dictionary file 3) All output files (**except test6out.txt** since it is very large) and 4) 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 May 31, 2019**. Note: Do **NOT submit the dictionary file or test6out.txt**.
Late Due Date: 11:59PM on Monday, June 3, 2019

W Students: Note your W exercise at the end of this document.

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. Programs that do not compile and run from the command line will be considered to be non-working solutions, even if they do run from your IDE.

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 **TrieSTNew** of these words. The TrieSTNew class is a modification of Sedgewick's TrieST class and can be found in file TrieSTNew.java. Carefully read over the code and comments in this class so that you understand what how the class works. **In particular, your anagram solution must utilize the searchPrefix() method** in order to prune the solution space. Use the file **dictionary.txt** on the CS1501 Assignments page.
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. It must also output the number of solutions found for each input string, and within that answer the number of solutions for each K where K is the number of words in a solution. **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:

– 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 (which is why you must use the searchPrefix() method). For example, using the input above ("raziber"), if we consider the letters in the order given, we can test in the following fashion (note each prefix corresponds to a recursive call):

   r | ra | raz | razi → backtrack at this point because "razi" is not a valid prefix in our word dictionary (the searchPrefix() method in the TrieSTNew object would return 0 for this string). Thus, we know a solution starting with "razi" is not possible and we can stop looking in the forward direction. Before moving forward again we should remove the "i" and try a different letter at that position. For example, in the case above we might try "razb", which is also not a prefix, then we might try "raze" which IS a prefix. At that point we can again move forward with our recursion. Note that before getting a solution we may end up having to backtrack all the way to the first character, as can be seen in this example. The two solutions that use all of the letters are "bizarre" and "brazier", so clearly our initial attempt that starts with "r" will not be successful.

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). In particular note that in this case if **a string is BOTH a prefix AND a word**, you will have to consider both of these possibilities in your recursive process. Don't forget to sort the solutions from fewest words to most words (and alphabetically for solutions with the same number of words) and don't forget to output the number of solutions overall and for each K where K is the number of words in a solution.

– If a letter (or letters) appear more than once in the initial string, your algorithm will likely produce some anagrams (possibly many) 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, so solutions with the same words in different positions should be kept. 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. See the Assignment 1 information on the Assignments web site for a listing.

– 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.

– **W Students: In addition to completing the project specified above, you must also write a short (~3-4 page, double-spaced) paper on the life and accomplishments of Donald Knuth. Your paper should cite at least 3 references that are NOT web sites (i.e. they must be books or legitimate published articles) and it should be well-written with a formal tone. In order to allow for comments / tracking of changes, you must use .docx as the format for this paper. Note that Pitt students should be able to obtain Microsoft Word free of charge. Submit your paper with your other submission materials in your .zip file.**