* First download a copy of the English dictionary that you will user from this project from <http://people.cs.pitt.edu/~nlf4/cs1501/handouts/dictionary.txt> and place thit in your repository folder
  + Do not add this file to your git repository, however! (i.e., do not run "git add dictionary.txt").
* You must implement a De La Briandais (DLB) trie data structure (as described in lecture) to use in your project.
* When your program is run, it should first create a new DLB trie and add all of the words in dictionary.txt to that trie (this will be the dictionary trie).
* Once the dictionary trie is established, you should prompt the user to start typing a word. For this project, you will be writing only a simple program to test autocomplete functionality. Due to complexities with gathering single character input in Java, you will simply be having the user enter a single character followed by "Enter" at a time and then present them with the list of predictions for what they are trying to type. If the user types a number (1-5), you should consider the user to have selected that prediction, and prompt them to enter another word. If the user enters a "$", you should consider the sequence of characters they have input up until this point to be the word they were trying to type (regardless of suggestions). If the user enters a "!", your program should exit.
* You program should propose 5 suggestions at a time.
* To generate the list of predictions, your program should not only consult the dictionary trie, but also keep track of what words the user has entered in the past. If the user has previously entered the same sequence of characters as a prefix to a word, you should should predict that they are trying to type the words that most frequently resulted from this sequence previously. If the user has never entered the current sequence before or has entered fewer than 5 words with the current seequence as a prefix (i.e., not enough words to complete the list of 5 predictions), your program should suggest words from dictionary.txt that have the current sequence as a prefix.
* The design of the data structure that keeps track of a user's previously entered words is entirely up to you. You must create a file named "approach.txt" that both describes your approach to implementing this symbol table and justifies your decision to take this approach. Note that this file does not need to be extensive, just a few lines so the TA is aware of what to look for in your code and why you chose this approach.
* This history of the user's entered words should persist across runs of your program. To accomodate this, your program should save a representation of this data structure to the file "user\_history.txt" before exiting.
  + Do not add this file to your git repository! (i.e., do not run "git add user\_history.txt") It should be generated by your program if it does not exist.
* If the current sequence of characters has not been entered by the user before and does not appear in dictionary.txt, you should display a message to the user stating that no predicions were found, and allow the user to continue entering characters one at a time. Once the user enters a "$", you should consider the word to be finished and add it to the user's history so that you can predict it in the future.
* Each time the user enters a character, you should use Java's System.nanoTime() (<https://docs.oracle.com/javase/8/docs/api/java/lang/System.html#nanoTime-->) to calculate how long your program takes to find the predictions. You should display this time along with the list of predictions.
* After the user enters "!", your program should output the average time that was required to produce a list of predictions.

An example run of the program would proceed as follows:

Enter your first character: t

(0.000251 s)

Predictions:

(1) t (2) ta (3) tab (4) tab's (5) tabbed

Enter the next character: h

(0.000159 s)

Predictions:

(1) thalami (2) thalamus (3) thalamus's (4) thalidomide (5) thalidomide's

Enter the next character: e

(0.000052 s)

Predictions:

(1) the (2) theater (3) theater's (4) theatergoer (5) theatergoer's

Enter the next character: r

(0.000225 s)

Predictions:

(1) therapeutic (2) therapeutically (3) therapeutics (4) therapeutics's (5) therapies

Enter the next character: e

(0.000182 s)

Predictions:

(1) there (2) there's (3) thereabout (4) thereabouts (5) thereafter

Enter the next character: 3

WORD COMPLETED: thereabout

Enter first character of the next word: t

(0.000128 s)

Predictions:

(1) thereabout (2) t (3) ta (4) tab (5) tab's (6) tabbed

Enter the next character: h

(0.000094 s)

Predictions:

(1) thereabout (2) thalami (3) thalamus (4) thalamus's (5) thalidomide (6) thalidomide's

Enter the next character: e

(0.000085 s)

Predictions:

(1) thereabout (2) the (3) theater (4) theater's (5) theatergoer (6) theatergoer's

Enter the next character: r

(0.000145 s)

Predictions:

(1) thereabout (2) therapeutic (3) therapeutically (4) therapeutics (5) therapeutics's (6) therapies

Enter the next character: e

(0.000130 s)

Predictions:

(1) thereabout (2) there (3) there's (4) thereabouts (5) thereafter

Enter the next character: !

Average time: 0.000145 s

Bye!

**Submission Guidelines:**

* **DO NOT** add "dictionary.txt" to your git repository.
* **DO NOT** add "user\_history.txt" to your git repository, it must be generated by your program.
* **DO NOT SUBMIT** any IDE package files.
* You must name your main program file ac\_test.java.
* You must be able to compile your program by running "javac ac\_test.java".
* You must be able to run your program by running "java ac\_test".
* You must fill out info\_sheet.txt.
* Be sure to remember to push the latest copy of your code back to your GitHub repository before the the assignment is due. At the deadline, the repositories will automatically be copied for grading. Whatever is present in your GitHub repository at that time will be considered your submission for this assignment.

**Additional Notes:**

* You are free to use any data structures written by you, the textbook authors, or in the Java standard library to implement the user history symbol table. However, if you use code that you do not write yourself, you must research the runtime of that particular implementation and discuss that in your approach.txt.
* Note that if your user history predictions contain a word that is also contained in the dictionary predictions, this word should not be presented as a suggestion to the user twice in the same prompt (e.g., the final list of predictions in the example above).
* You do not need to implement any sort of autocorrect. You can assume that each character entered by the user is intentional.
* You can assume that the user will not try to enter and numerical characters aside from selecting a prediction.