Due : Nov. 28 (11/28), 13:00
Late submission due: Nov. 29 (11/29), 13:00

Overview

This assignment contains two part.

General Notes

- *Read this homework guideline carefully.* If you do not follow the guidelines, you may receive a 0 regardless of whether your code works or not.
- Do not use any IDEs (Eclipse, IntelliJ IDEA, etc.)
  - We recommend Sublime Text (Linux/Mac/Windows), Atom (Linux /Mac/Windows), Notepad++ (Windows), or TextWrangler (Mac).
  - IDEs often create a "package" of your code, which breaks the autograder.
  - **If you know how to fix the package problem**, you can use any IDE you want. However, we will not answer any questions related to this problem since we have already recommended a solution.
- Do not change any method or class signatures. If your code changes any class or method names or signatures, you will receive an automatic 0.
- Make sure your code compiles. Non-compiling code will automatically receive a 0. If you have a problem that is causing you to not be able to compile, it may be better to just comment out the incorrect code and return a dummy value (something like null or -1) so the rest can compile.
- To ensure that your code will be accepted by the autograder, you should submit your code on YSCEC, download it again, recompile it and check the provided test suite. This way, you know that the file you are submitting is the correct one.
- You can use any course materials. However, if you do not cite the source you referred to, it might be checked as copied code. Please write the material (and page) you referred in comments.

1 Trie

In computer science, a trie, also called digital tree or prefix tree, is a kind of search tree; an ordered tree data structure used to store a dynamic set or associative array where the keys are usually strings.

1.1 Trie

In this assignment, you will implement the `standard` trie in Trie.java. You can use the Node.java that stores the children of the node and flag whether the current node is the leaf of the tree. Also, for each node in the trie, its children must be stored in lexicographic order.

You must implement the following methods as well as its constructor:

`insert`: Insert the given word into the trie.

`exists`: Return `true` if the given word exists in the trie. Otherwise, return `false`.

`dictionary`: Return all words from the trie in lexicographic order.

`root`: Return the root node of the trie.

1.2 Autocomplete

Most search engines have a feature that automatically completes the search word you enter. In this problem you will implement Autocomplete.java, a program that automatically completes the given input stream and calculates the average number of input counts needed. The operation process of autocomplete is as follows:

1. The program cannot guess the first character of a word, so the first character has to be given as input.
2. Once the user inputs a sequence of characters $c_1c_2 \cdots c_n$ ($n \geq 1$), and there is a character c such that every word in the dictionary which starts with $c_1c_2 \cdots c_n$ is followed by the character c ($c_1c_2 \cdots c_nc$), then the program inputs c automatically without the need of input.
3. Otherwise, the system waits for user input.

For example, If the dictionary is composed of the words "yonsei", "young", "yoga", "youtube", and "datastructure" and the user input is "y", the program will return "yo" automatically, because every word which starts with "y" also starts with "yo". The next input can be "n" then the program return "yonsei" since the "yonsei" is the only word that starts with "yon" in the dictionary. Similarly, when the user inputs a "d", the program returns "datastructure" with a single input, as there is only one matching word in the dictionary.

Therefore, the program requires 2 times of input to autocomplete the word "yonsei", and 3 times of input each to autocomplete "young" and "youtube", 2 times of input to autocomplete "yoga" and 1 time of input to autocomplete "datastructure". So the average number of inputs the program needs to autocomplete the words is (2 + 3 + 3 + 2 + 1) / 5 = 2.2.

You must implement the following methods as well as its constructor in the autocomplete.java file:

`construct`: Given an array of words, store the given words to make a dictionary.

`autocompletedWord`: Return the current autocompleted word for the given input.

`average`: Return the average number of inputs for the words in the dictionary. You should round the result to the second digit after the decimal point.

You have to return autocompleted word in $O(1)$ time.

General Directions

- Write your name and student ID number in the comment at the top of the files you submit.
- Implement all of the required methods.
- You should not import anything that is not already included in the file.
- Pay careful attention to the required return type and edge cases.
- All the codes we provide can be found in src/base directory. If you are unsure what a class/method exactly does, please refer to the code.
- You are free to implement any algorithm that you wish, but you must code it yourself. If you referred to any course materials, you must write the name of the material and page in comments. We will only be testing that your code produces a correct result and terminates in a reasonable amount of time.

Submission Procedure

You *must* make a zip file for submission using Gradle build tool (refer to Compiling section). For this assignment, the zip file will contain only the following three files:

- Trie.java
- Autocomplete.java
- your_student_id_number.txt

You must rename 2020xxxxxx.txt to your actual student ID number. Inside of that text file, you must include the following text and write your name at the bottom. Please be sure to write all the following text including the last period.

In completing this assignment, I pledge that I have not given nor received any unauthorized assistance.

If this file is missing, you will get 0 on the assignment. It should be named *exactly* your student id, with no other text. For example, *2020123456.txt* is correct while something like *2020123456_pa7.txt* will receive 0.

Compiling

This assignment uses Gradle build tool to automate compiling and testing procedure. The following command will test your Java code against the provided testcases:

```
% ./gradlew -q runTestRunner
```

The following command will zip the files for your submission. The zip file will be named with your student id (the name of .txt file) and will lie in "build" directory. Be aware that the command will be interrupted if your pledge does not comply the guideline.

```
% ./gradlew -q zipSubmission
```

Since the testrunner blocks the standard output from printing, it is hard to test your code fragment while writing the code. For this purpose, we also provide an empty Main class. As this file is not for submission, you may use any features Java provides. The following command will run the Main class instead of the testcases.

```
% ./gradlew -q runMain
```

On Windows, try `gradlew.bat` instead of `./gradlew` if you met an error. Moreover, you may omit the '`-q`' option to review the compile log.

Testing

We have provided small testcases (src/test) for you to check your code. You can test your code by the means mentioned above.

Note that the testcases we will use to grade your code is `very much more` rigorous than the one provided here (and not necessarily a superset of the provided tests). You should consider making your own test cases to check your code more thoroughly.