Project 3: Strie

DUE: Sunday, November 3rd at 11:59pm

Extra Credit Available for Early Submissions!

Basic Procedures

You must:

- Fill out a `readme.txt` file with your information (goes in your user folder, an example is provided)
- Have a style (indentation, good variable names, etc.)
- Comment your code well in JavaDoc style (no need to overdo it, just do it well)
- Have code that compiles with the command: `javac *.java` in your user directory
- Have code that runs with the command: `java StrieDemo`

You may:

- Add additional methods and variables, however these **must be private**.
- Use the `java.util`'s `ArrayList` locally in `Strie` methods. You **may not** have an instance variable of `ArrayList` and you **may not** use it in any classes other than `Strie`.

You may NOT:

- Make your program part of a package.
- Add additional public methods or variables
- Use any built in Java Collections Framework classes anywhere in your program other than in the ways listed in the "you may" section (e.g. no `LinkedList`, `HashSet`, etc.). Required classes are all provided in the given templates.
- Alter any method signatures defined in this document of the template code. Note: "throws" is part of the method signature in Java, don't add/remove these.
- Alter provided classes/methods that are complete (`StrieDemo`, `main()`, etc.).
- Add any additional import statements (or use the "fully qualified name" to get around adding import statements).
- Add any additional libraries/packages which require downloading from the internet.

Setup

- Download the `project3.zip` and unzip it. This will create a folder `section-yourGMUUserName-p3`
- Rename the folder replacing `section` with the `001`, `003`, `004` etc. based on the lecture section you are in
- Rename the folder replacing `yourGMUUserName` with the first part of your GMU email address
- After renaming, your folder should be named something like: `001-krusselc-p3`
- Complete the `readme.txt` file (an example file is included: `exampleReadmeFile.txt`)

Submission Instructions

- Make a backup copy of your user folder!
- Remove all test files, jar files, class files, etc.
- You should just submit your java files and your readme.txt
- Zip your user folder (not just the files) and name the zip `section-username-p3.zip` (no other type of archive) following the same rules for `section` and `username` as described above.
    - The submitted file should look something like this:

```
001-krusselc-p3.zip --> 001-krusselc-p3 --> JavaFile1.java
                                            JavaFile2.java
                                            ...
```

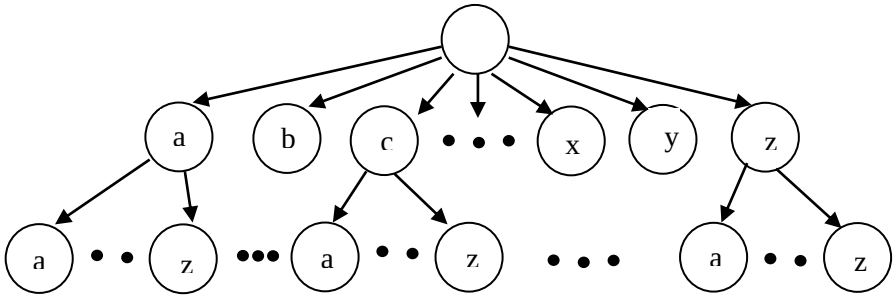
- Submit to blackboard.

Grading Rubric

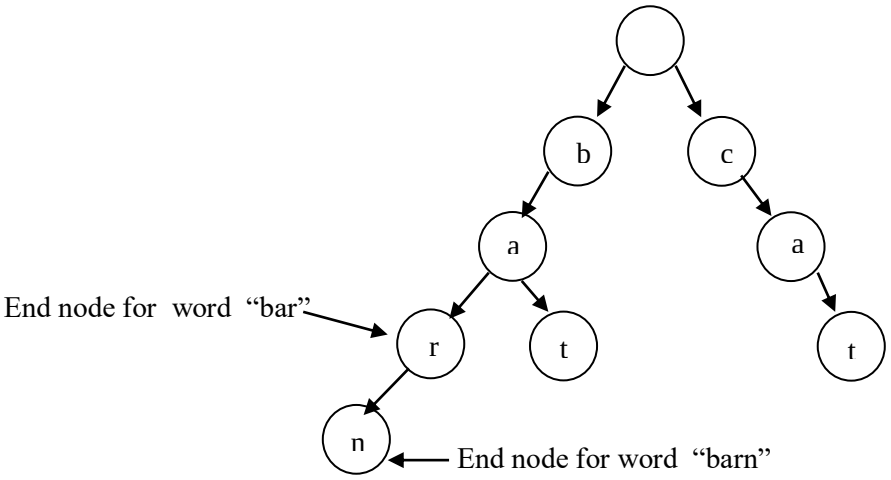
Due to the complexity of this assignment, an accompanying grading rubric pdf has been included with this assignment. Please refer to this document for a complete explanation of the grading.

Overview: Storing a Dictionary with a Tree

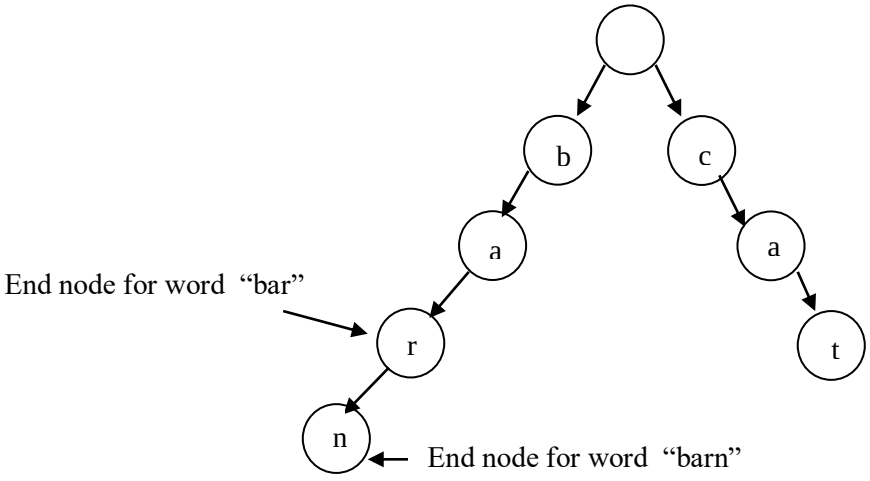
Strie is a tree data structure that stores a set of words in a very space efficient way. The following is an example of basic structures of a Strie and Strie nodes.



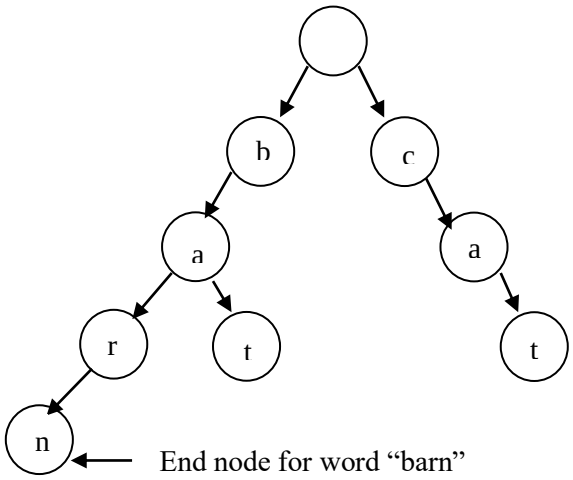
To provide this unique capability of storing character strings, Stries need to follow some rules. Each node of a Strie, except the root node, contains a character and can have up to 26 children. Root node (and all other nodes) store(s) links to other nodes, one for each possible character in the English alphabet. Following is a sample Strie that has 4 words (bar, barn, bat, cat) stored in it.



If you look carefully at the above Stries, you would notice some interesting properties that may help you in implementing this project! For example, since there are 26 possible slots for the children of a node and since the alphabet is in order, you will always find a particular character at a particular index! Also, words with common prefix share a common branch! Note that words sharing common prefixes may affect the implementation of some operations. For example, if you remove the word "bat", your Strie would look like the following picture.



The character 't' does not exist in the Strie anymore under 'ba' but it does still exist under 'ca' for 'cat'. In contrast, if you remove the word "bar", your Strie would look like following picture. Notice that the character "r" still exists in Strie, but it no longer marks the end of a word.



You can use the Strie for many different purposes. For example, you can store words, look up a word, search for words having common prefix, search for similar words for a given query, and many more. To implement such a Strie, you need to complete all required classes, and possibly some (or all) optional classes depending on which functionality you want your Strie to provide.

Requirements

An overview of the requirements are listed below, please see the grading rubric for more details.

- **Implementing the classes** - You will need to implement required classes in the provided template files.
- **Big-O** - The template files provided to you contains instructions on the REQUIRED Big-O runtime for many methods. Your implementation of those methods should NOT have a higher Big-O.
- **Style** – You must follow the coding and conventions specified by the style checker.
- **JavaDocs** - You are required to write JavaDoc comments for all the required classes and methods.

Implementation/Classes

This project will be built using a number of classes representing the component pieces of the project. Here we provide a description of these classes. Template files are provided for each class in the project folder and these contain further comments and additional details.

Completed Class

- **The `StrieDemo` Class (see StrieDemo.java)** This class is done for you and you can use it to interact with a Strie you've created.

Required Classes

- **The `StrieNode` Class (see StrieNode.java)** This class represents a single Strie node. It provides some methods required to manipulate a node. Check the included template file for all required methods and their corresponding big-O requirements.

- **The `Strie` Class (see Strie.java)** This class implements the Strie data structure. It must use `StrieNode` objects for storing a node. It supports some basic Strie operations such as inserting a word, removing a word, searching for a word, and more.

    - **IMPORTANT:** Not all the methods in this class are required (some are optional), but you cannot remove any of them from the template. If you remove any of the given method signatures, your code may not compile with the grading tests.

- **Required methods:**
  - All methods in **StrieNode** class
  - **Strie** class methods:
    - `insert(String word)`: inserts a word in Strie
    - `remove(String word)`: removes a word from Strie
    - `contains(String word)`: search the Strie for word
    - `levelOrderTraversal(Strie myStrie)`: do a breadth first traversal
    - `isEmptyStrie(Strie myStrie)`: check if your Strie is Empty
  - In summary, all methods except the ones mentioned below are required.
- **Choose your own adventure methods:**
  - Implement one (or all) of the following methods. There is a maximum number of points you can earn with these methods (please see the Grading Rubric) no matter how many methods you implement. These methods are each fairly complex, but provide you with the practice you'll need to really understand this data structure.
    - `getStrieWords(Strie myStrie)`: get all existing words in your Strie
    - `getLongestPrefix(Strie myStrie, String query)`: get from Strie the longest prefix for a given query
    - `getAllSuffixes(Strie myStrie, String query)`: suggest all words from your Strie for a given prefix (i.e., query)
    - `getClosestMatch(Strie myStrie, String query)`: get from your Strie the closest match for a given word

How To Handle a Multi-Week Project

This project is given to you to work on over two weeks. You are unlikely to be able to complete this in one weekend if you don't plan ahead. We recommend the following schedule:

- Step 1 (Before the second weekend, up to October 18th) – Setup:
  - Read project specification, inspect the given template files.
  - Get yourself familiar with the **Strie** data structure. Practice making Stries by hand, inserting, removing, and searching for words.
  - Implement the **StrieNode** class and one or more required methods of **Strie** class, if possible.
- Step 2 (Second weekend and following week, Oct 19th-27th) – Required Strie Methods:
  - Complete all the required methods of **Strie** class.
  - Test and document as you go!
- Step 3 (Third weekend and following week, Oct 28th to Nov 3rd) – Chosen Strie Methods:
  - Choose and complete one or more of the remaining methods of **Strie** class
  - Test the **Strie** class with as many inputs as you can!

Notice that during the last week, if you submit early you can get extra credit! Check our grading rubric PDF for details. Otherwise you'll have plenty of time to test and debug your implementation before the due date.

Testing

- Some tests are provided for you in the main method of **Strie** class.
- You can run the simulation StrieDemo once everything is done.
  - Note: Testing via the simulation *only* is not a good idea. Some of the methods you need to write are not used by the simulation, but are still a graded part of the project.
- Make sure you're running the code style checker and JavaDoc style checker.