HW 11: Tries

You must create a `Trie` object, which is used to store a very large set of `String`s with ultrafast constant-time access. You will also need to create a class called `WordChecker3`, that makes use of the `Trie`. Note that unlike the `BinarySearchTree` you made last week, a `Trie` works only with `String`s, and hence will not need to use generic types.

Like in this week's lab, the first thing your program will do is to ask the user for a file. If this file is a `.lex` file, it will load all the words in that file into a brand new `Trie`, state the number of words, and save it to disk. The filename it uses should be the same as the input file, with the `.lex` extension replaced by `.trie`:

```
Welcome to the Word Checker 3.0!
Please enter a .lex or .trie file:  english.lex
Loading file "english.lex", which contains 62970 words.
Saving new file "english.trie"...done.
Goodbye!
```

If the file the user enters is a `.trie` file, you will load it up, state how many words the Trie contains, and enter query mode as in previous assignments.

Your implementation must include the following functions:

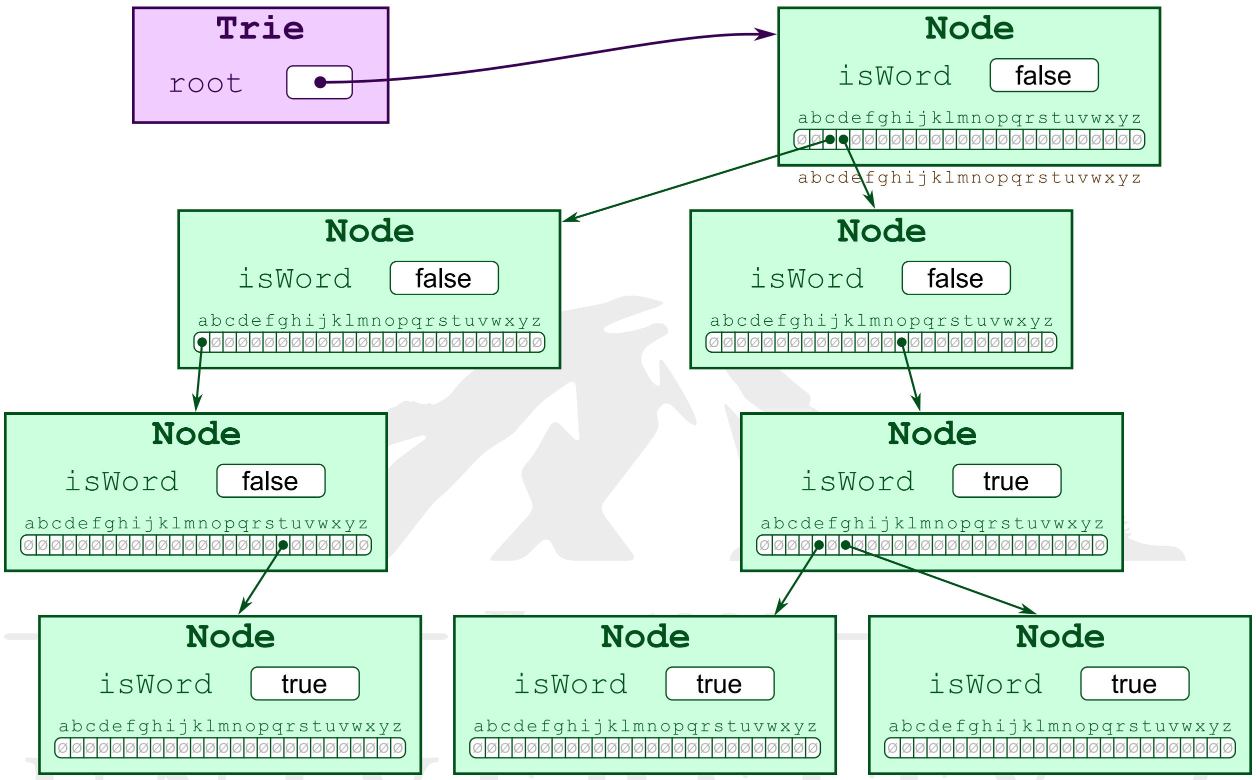
- `Trie()`, which makes a brand new `Trie`. It should be initialized so that its root holds a single `Node`.
- `void insert(String word)`, which inserts a new word into the `Trie`.
- `boolean has(String word)`, which returns `true` iff the given value is present in the `Trie`.
- `int getSize()`, which returns the number of words inside this `Trie`. (You will probably want to maintain a counter.)

Each of these functions should have time complexity of O(1) with respect to the number of words the `Trie` contains.

Note that for the first time, you'll be saving an object to disk that you created yourself. Thus, as we went over in class, your `Trie` must contain a private static long called `serialVersionUID`, that holds some special 18-digit value that is unique to you. The same is true for the nested `Node` object—and its `serialVersionUID` must hold a different value.

Some hints:

- While you are working on this assignment, make your own `.lex` file that contains only 10 words or so. That way you can test it with a much more human-sized data set. When you think it works, try the full file.
- You may want to create your own temporary `print()` function which prints out the `Trie`. This would be solely for your own debugging purposes.



- There is no need to shuffle the words, as there was for the binary search tree.

Also, when you have finished this assignment, take a look at the size of the `.trie` file that was created. Compare it to the size of the `.set` file in this weeks lab. A trie is faster, but that speed comes at a price!

For extra credit, you may create the following methods:

- `String findMin()`, which returns the first word (alphabetically) in this `Trie`. Your query system should call this function if the user enters `*first`.
- `String findMax()`, which returns the last word (alphabetically) in this `Trie`. Your query system should call this function if the user enters `*last`.
- `boolean delete(String word)`, which deletes this word and returns `true` if it was in the tree, or just returns `false` if it was not. Any extra `Node`s should be delted. Your query system should test this if the user enters `-word`, where `word` is the word to be delted. If it was not present, state this and continue.

Implementing this functionality can add as much as 25% to your grade.