Project 3 (Autocomplete Me)

Clarifications and Hints

Prologue

Project goal: write a program to implement *autocomplete* for a given set of N strings and nonnegative weights, ie, given a prefix, find all strings in the set that start with the prefix, in descending order of weight

The zip file (`http://www.swamiiyer.net/cs210/project3.zip`) for the project contains

- project specification (`project3.pdf`)
- starter files (`Term.java`, `BinarySearchDeluxe.java`, `Autocomplete.java`)
- test script (`run_tests.py`)
- test data (`data/`)
- visualization client (`AutocompleteGUI`)
- report template (`report.txt`)

This checklist will help only if you have read the writeup for the project and have a good understanding of the problems involved. So, please read the project writeup before you continue with this checklist.

Problems

Problem 1 (*Autocomplete Term*) Implement an immutable comparable data type `Term` that represents an autocomplete term and has the following API:

method	description
`Term(String query)`	initialize a term with the given query string and zero weight
`Term(String query, long weight)`	initialize a term with the given query string and weight
`static Comparator<Term> byReverseWeightOrder()`	compare the terms in descending order by weight
`static Comparator<Term> byPrefixOrder(int r)`	compare the terms in lexicographic order but using only the first r characters of each query
`int compareTo(Term that)`	compare the terms in lexicographic order by query
`String toString()`	a string representation of the term

Hints

- Instance variables
  - Query string, `String query`
  - Query weight, `long weight`
- `Term(String query)` and `Term(String query, long weight)`
  - Initialize instance variables to appropriate values

Problems

- `static Comparator<Term> byReverseWeightOrder()`
  - Return an object of type `ReverseWeightOrder`
- `ReverseWeightOrder.compare(Term v, Term w)`
  - Return a -1, 0, or +1 based on whether `v.weight` is smaller, equal to, or larger than `w.weight`
- `static Comparator<Term> byPrefixOrder(int r)`
  - Return an object of type `PrefixOrder`
- Instance variable for `PrefixOrder`
  - Prefix length, `int r`
- `PrefixOrder(int r)`
  - Initialize instance variable appropriately
- `PrefixOrder.compare(Term v, Term w)`
  - Return a negative, zero, or positive integer based on whether `a` is smaller, equal to, or larger than `b`, where `a` is a substring of `v` of length `min(r, v.query.length())` and `b` is a substring of `w` of length `min(r, w.query.length())`
- `int compareTo(Term that)`
  - Return a negative, zero, or positive integer based on whether `this.query` is smaller, equal to, or larger than `that.query`

Problems

Problem 2 (*Binary Search Deluxe*) Implement a library of static methods `BinarySearchDeluxe` with the following API:

method	description
`static <Key> int firstIndexOf(Key[] a, Key key, Comparator<Key> comparator)`	the index of the first key in $a[]$ that equals the search key, or -1 if no such key
`static <Key> int lastIndexOf(Key[] a, Key key, Comparator<Key> comparator)`	the index of the last key in $a[]$ that equals the search key, or -1 if no such key

Hints

- `static int firstIndexOf(Key[] a, Key key, Comparator<Key> comparator)`
  - Modify the standard binary search such that when `a[mid]` matches `key`, instead of returning `mid`, remember it in, say `index` (initialized to -1), and adjust `hi` appropriately
  - Return `index`
- `static int lastIndexOf(Key[] a, Key key, Comparator<Key> comparator)` can be implemented similarly

Problem 3 (*Autocomplete*) Create an immutable data type `Autocomplete` with the following API:

method	description
`Autocomplete(Term[] terms)`	initialize the data structure from the given array of terms
`Term[] allMatches(String prefix)`	all terms that start with the given prefix, in descending order of weight
`int numberOfMatches(String prefix)`	the number of terms that start with the given prefix

Hints

- Instance variable
  - Array of terms, `Term[] terms`
- `Autocomplete(Term[] terms)`
  - Make a defensive copy of `terms` into `this.terms`
  - Sort `terms` in lexicographic order

Problems

- `Term[] allMatches(String prefix)`
  - Use `BinarySearchDeluxe` and `Term.byPrefixOrder()` to obtain the first index `i` of occurrence of `prefix`
  - Find the number `n` of terms that match `prefix`
  - Construct an array `matches` containing `n` elements from `terms`, starting at index `i`
  - Sort `matches` in reverse order of weight and return the sorted array
- `int numberOfMatches(String prefix)`
  - Use `BinarySearchDeluxe` and `Term.byPrefixOrder()` to obtain the first index and last index of occurrence of `prefix`
  - Compute and return the number of terms that match `prefix`

Epilogue

The `data` directory contains sample input files for testing; for example

```
$ more data/wiktionary.txt
10000
   5627187200   the
   3395006400   of
   2994418400   and
   2595609600   to
   1742063600   in
   1176479700   i
   1107331800   that
   1007824500   was
    879975500   his
                ...
       392323   calves
```

The visualization client `AutocompleteGUI` takes the name of a file and an integer k as command-line arguments, provides a GUI for the user to enter queries, and presents the top k matching terms in real time

Epilogue

To receive full credit for a problem, your solution must implement the "corner cases" (if any) and meet the "performance requirements" (if any)

Your project report (use the given template, `report.txt`) must include

- time (in hours) spent on the project
- short description of how you approached each problem, issues you encountered, and how you resolved those issues
- acknowledgement of any help you received
- other comments (what you learned from the project, whether or not you enjoyed working on it, etc.)

Before you submit your files

- make sure your programs meet the input and output specifications by running the following command on the terminal

  ```
  $ python3 run_tests.py -v [<problems>]
  ```

  where the optional argument `<problems>` lists the problems (`Problem1`, `Problem2`, etc.) you want to test, separated by spaces; all the problems are tested if no argument is given
- make sure your programs meet the style requirements by running the following command on the terminal

  ```
  $ check_style <program>
  ```

  where `<program>` is the `.java` file whose style you want to check
- make sure your report isn't too verbose, doesn't contain lines that exceed 80 characters, and doesn't contain spelling/grammatical mistakes