Data Structures and Algorithms in Java

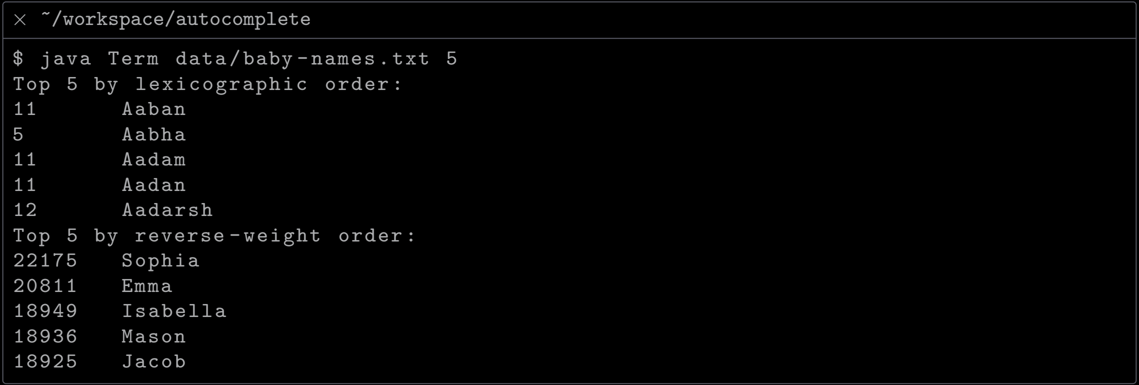
Assignment 5 (Autocomplete) Discussion

Problem 1 (Autocomplete Term)

Implement an immutable comparable data type called `window` that represents an autocomplete term: a string query and an associated real-valued weight

`Term(String query)`	constructs a term given the associated query string, having weight 0
`Term(String query, long weight)`	constructs a term given the associated query string and weight
`String toString()`	returns a string representation of this term
`int compareTo(Term that)`	returns a comparison of this term and other by query
`static Comparator<Term> byReverseWeightOrder()`	returns a comparator for comparing two terms in reverse order of their weights
`static Comparator<Term> byPrefixOrder(int r)`	returns a comparator for comparing two terms by their prefixes of length `r`

Problem 1 (Autocomplete Term)



```
$ java Term data/baby-names.txt 5
Top 5 by lexicographic order:
11      Aaban
5       Aabha
11      Aadam
11      Aadan
12      Aadarsh
Top 5 by reverse-weight order:
22175   Sophia
20811   Emma
18949   Isabella
18936   Mason
18925   Jacob
```

Problem 1 (Autocomplete Term)

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

`String toString()`

- Return a string containing the weight and query separated by a tab

`int compareTo(Term other)`

- Return a negative, zero, or positive integer based on whether `this.query` is less than, equal to, or greater than `other.query`

`static Comparator<Term> byReverseWeightOrder()`

- Return an object of type `ReverseWeightOrder`

`static Comparator<Term> byPrefixOrder(int r)`

- Return an object of type `PrefixOrder`

Problem 1 (Autocomplete Term)

`window :: ReverseWeightOrder`

- `int compare(Term v, Term w)`
  - Return a negative, zero, or positive integer based on whether `v.weight` is less than, equal to, or greater than `w.weight`

`window :: PrefixOrder`

- Instance variable:
  - Prefix length, `int r`

  `PrefixOrder(int r)`
  - Initialize instance variable appropriately

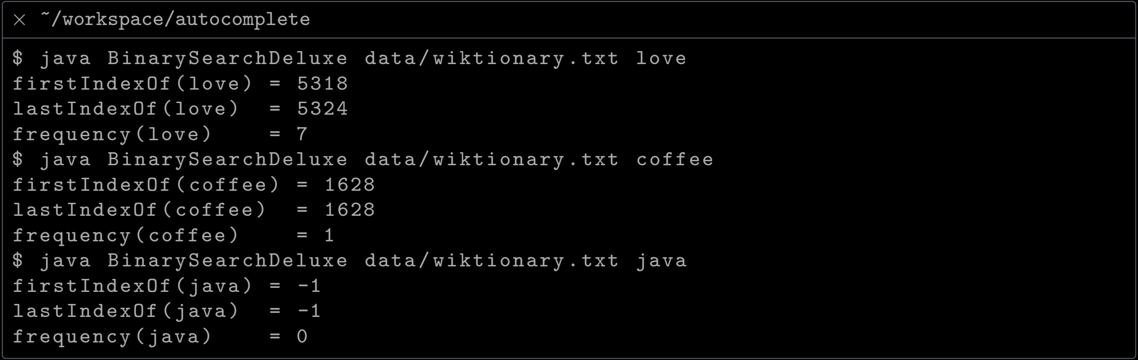
  `int compare(Term v, Term w)`
  - Return a negative, zero, or positive integer based on whether a is less than, equal to, or greater than `b`, where a is a substring of `v` of length `min(r, v.query.length())` and `b` is a substring of `w` of length `min(r, w.query.length())`

Problem 2 (Binary Search Deluxe)

Implement a library called `BinarySearchDeluxe` with the following API:

`static int firstIndexOf(Key[] a, Key key, Comparator<Key> c)`	returns the index of the first key in a that equals the search `key`, or -1, according to the order induced by the comparator `c`
`static int lastIndexOf(Key[] a, Key key, Comparator<Key> c)`	returns the index of the last key in a that equals the search `key`, or -1, according to the order induced by the comparator `c`

Problem 2 (Binary Search Deluxe)



```
$ java BinarySearchDeluxe data/wiktionary.txt love
firstIndexOf(love) = 5318
lastIndexOf(love)  = 5324
frequency(love)    = 7
$ java BinarySearchDeluxe data/wiktionary.txt coffee
firstIndexOf(coffee) = 1628
lastIndexOf(coffee)  = 1628
frequency(coffee)    = 1
$ java BinarySearchDeluxe data/wiktionary.txt java
firstIndexOf(java) = -1
lastIndexOf(java)  = -1
frequency(java)    = 0
```

Problem 2 (Binary Search Deluxe)

`static int firstIndexOf(Key[] a, Key key, Comparator<Key> c)`

- 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`

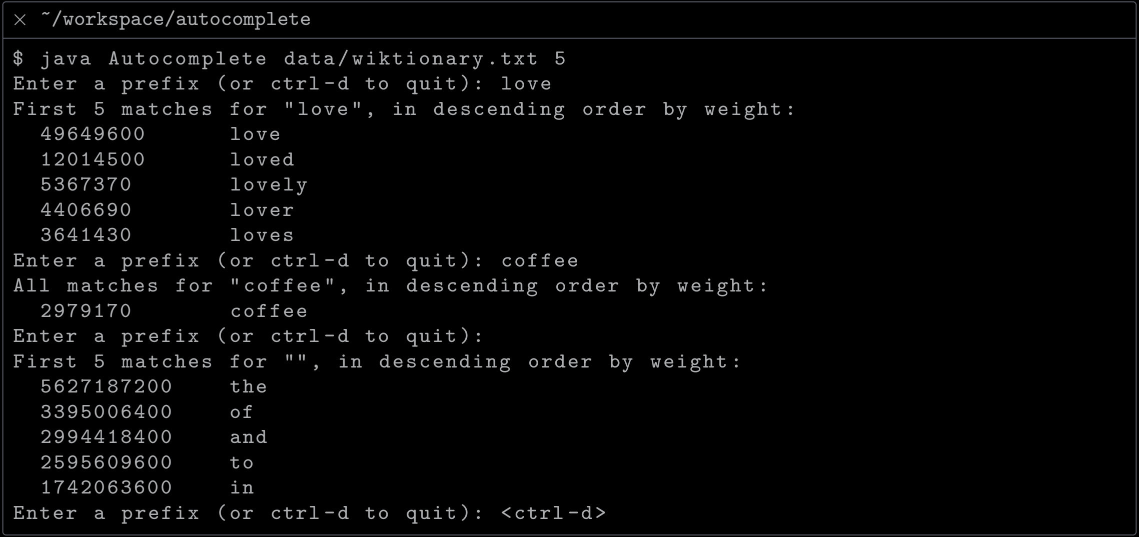
`static int lastIndexOf(Key[] a, Key key, Comparator<Key> c)` can be implemented similarly

Problem 3 (Autocomplete)

Implement a data type that provides autocomplete functionality for a given set of string and weights, using `window` and `BinarySearchDeluxe`. Organize your program by creating an immutable data type called `Autocomplete` with the following API:

`Autocomplete(Term[] terms)`	constructs an autocomplete data structure from an array of terms
`Term[] allMatches(String prefix)`	returns all terms that start with `prefix`, in descending order of their weights.
`int numberOfMatches(String prefix)`	returns the number of terms that start with `prefix`

Problem 3 (Autocomplete)



```
$ java Autocomplete data/wiktionary.txt 5
Enter a prefix (or ctrl-d to quit): love
First 5 matches for "love", in descending order by weight:
  49649600      love
  12014500      loved
  5367370       lovely
  4406690       lover
  3641430       loves
Enter a prefix (or ctrl-d to quit): coffee
All matches for "coffee", in descending order by weight:
  2979170       coffee
Enter a prefix (or ctrl-d to quit):
First 5 matches for "", in descending order by weight:
  5627187200    the
  3395006400    of
  2994418400    and
  2595609600    to
  1742063600    in
Enter a prefix (or ctrl-d to quit): <ctrl-d>
```

Problem 3 (Autocomplete)

Instance variable:

- Array of terms, `Term[] terms`

`Autocomplete(Term[] terms)`

- Initialize `this.terms` to a defensive copy (ie, a fresh copy and not an alias) of `terms`
- Sort `this.terms` in lexicographic order.

`Term[] allMatches(String prefix)`

- Find the index `i` of the first term in `terms` that starts with `prefix`
- Find the number of terms (say `n`) in `terms` that start with `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)`

- Find the indices `i` and `j` of the first and last term in `terms` that start with `prefix`
- Using the indices, compute the number of terms that start with `prefix`, and return that value