CS1006 W04Practical 170005030, 160005106 Xu Zhu 21th February, 2018

Overview and Aim:

In this practical, we were asked to write, compile and run a JAVA based front-end UI application which would take in user input and do a predictive autocomplete based on lexicographical order of N words in a text file which were sorted in their terms of weights. Given a particular prefix word – an autocomplete was implemented based on the query input and associated weight.

Design of Program:

The GUI made use of eight major java scripts – each implementing a different functionality. The front-end was built using JavaSwing and making use of external packages like key-listener/event (for event detection when a particular term is typed and the graphics were designed using JPanel, Frame and GUI buttons.

1. Term.java
   This script takes in two arguments – query, associated weight and implements a ReverseWeightOrderComparator() from a different script which arranges the terms on the basis magnitudes of weight values. Next – a PrefixOrderComparator() from a different script which is used to compare the length of the query string and the input string and returns the length of the query string if it is greater than the input string and vice versa which is then used in extracting matches based on certain prefix order words. This script also makes use of getter, setter methods to get weights and query's and uses a compareTo() to compare all set of query terms.

2. QueryHeapSort.java and WeightHeapSort.java

   These two scripts add the actually logic to the sorting of the terms. A heap sort algorithm with has a time complexity of n Log n. The sorting takes place for both the query's and the associated weights. The sorting algorithm has a run time of 136 ms for sorting all the terms of 'cities.txt' file and 29 ms for 'wiktionary.txt' file.

3. BinarySearchDeluxe.java

   The input of the binary search is an ordered array of all the queries from the .txt file. The binary search helps in getting the first and the last index of a particular key. The input of the binary search is halved on every run and the search compares the index of the first half and/or the second half based on the user input – (compare method).

4. Autcomplete.java

   In this script all the logic comes together in order. First the heap sort on queries is implemented on the query terms, next, the sorted queries are inputted in the binary search algorithm for first and last indices. Then, heap sorting is computed on the associated weights of the queries to get descending order of queries (for a term) based on weights.

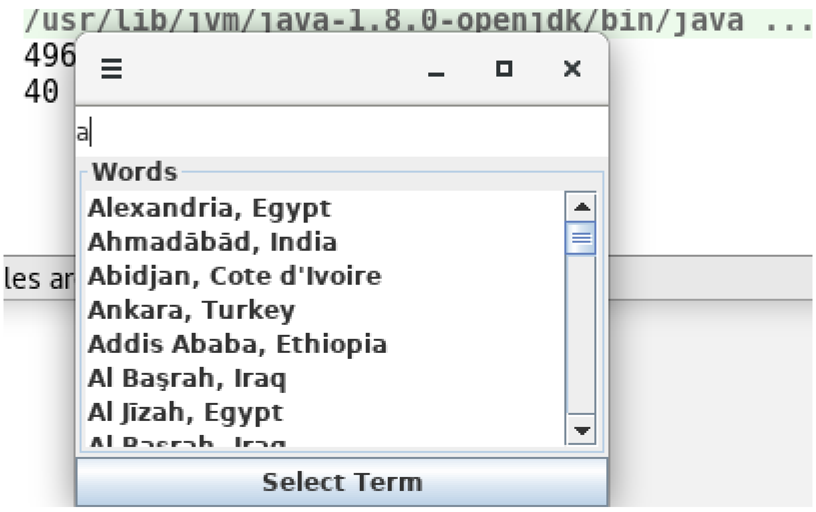
5. W04Practical.java

   In this script, the front-end GUI is developed using JSwing. A JScrollPane package is used for scrolled autocomplete words for a given user input. A keyListener and ActionListener events are used to check the user input after every key press. A button is used on the JPanel to select the autocomplete term. On an average, the UI returns a list of suggestions in less than 40 milliseconds.

Testing of Program and Output:

The code compiled and ran successfully and almost all possible situations worked as desired. We tested that the terms were arranged in descending order of weights lexicographically based on user input.

Sample Output (Cities.txt): for alphabet ‘a’:

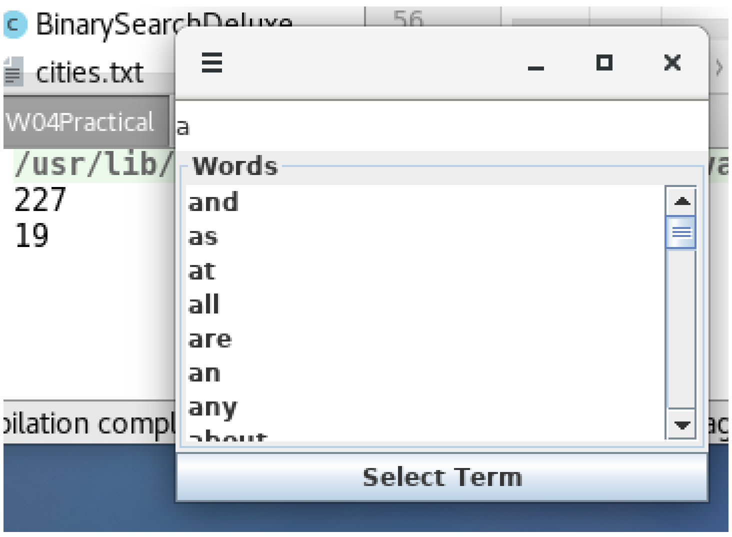


```
/usr/lib/jvm/java-1.8.0-openjdk/bin/java ...
496
40
```

a

Words
Alexandria, Egypt
Ahmadābād, India
Abidjan, Cote d'Ivoire
Ankara, Turkey
Addis Ababa, Ethiopia
Al Başrah, Iraq
Al Jīzah, Egypt

Sample Output (Wikitionary.txt) for term ‘a’:



```
227
19
```

a

Words
and
as
at
all
are
an
any

Link to GitHub private Repo:

https://github.com/Rohan2821999/CS1006_Autocomplete

Personal Contribution 160005106

<u>Term Class (PrefixOrderComparator and ReverseWeightOrderComparator)</u>

First thing I had to do was make the class, which can save the data (query and weight). Term class implement the Comparable interface that allows the programme to compare the weight in lexicographical order (by adding compareTo method). For the constructor, there is if statement. This statement throws the IllegalArgumentException if a query and a weight are null. Also, there are two return methods which return Comparator. The first method loads ReverseWeightOrderComparator class. This class implements Comparator interface to compare the weights to Term class. Another method is byPrefixOrder with parameter r (r is the length of prefix). This method also loads PrefixOrderComparator. PrefixOrderComparator compares two terms by given prefix.

<u>QueryHeapSort & WeightHeapSort (Extension)</u>

Both two classes are using for the heap sorting. WeightHeapSort uses upheap to make unsorted data to be descending order. UpHeap is the process of inserting a new element into the heap and constructing a new heap with the new element. The process is to place the newly added element at the farthest end compare the parent node with the value, and exchange the position if the value is greater than the parent node. This process repeats itself until it is located at the root node or is smaller than the parent node in the heapsort method. QueryHeapSort uses downheap which makes the unsorted data to be ascending order.

<u>BinarySearchDeluxe Class</u>

This class is for searching the word in the given (sorted) data. The integer firstIndex and lastIndex are initialized in -1 because, if there are no such key, it has to return -1. There are two methods one is firstIndexOf another is lastIndexOf. The firstindexof method will keep searching the key until it finds the first index of given prefix and same logic is applied in the lastIndexOf method.

Link to Mercurial

https://wl39.hg.cs.st-andrews.ac.uk/myrepos/