Data Structures and Algorithms : Predictive Text Assignment

Download the partial solution from moodle. Please study the code and make sure you understand how the system works before you start to develop your solution. You must be confident that what you have done is correct before you move on to the next question. Submit your final source code via Moodle.

Background

Most mobile phones offer predictive text to make easier enter messages write for instance text with mobile phone keyboards, which usually only contain 12 keys. Several letters are then written pressing the same key.

Predictive text allows the user to press a key only once to reach any of the letters it represents. A dictionary is used by the system to "predict" which of the letters of the key the user actually wanted to use. The principle relies on the fact that in a language, a sequence of keystrokes only corresponds to few words.

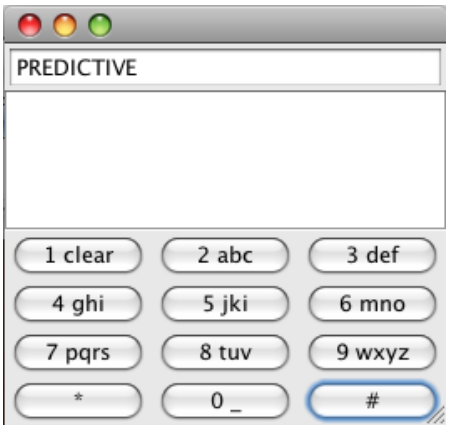

Figure 1: Screenshot of Java Mobile Phone Interface

As can be seen in figure 1, this project will make use of a mock up of a mobile phone interface, written in Java. Pressing 1 results in the last keystroke being removed, pressing 2-9 results in a character being added to the word that is currently being entered, pressing 0 inserts a space (that acts as a separator between words), pressing # switches between predictive text mode and standard text mode (where pressing a key repeatedly in quick succession causes the current character to cycle through its associated options), and pressing * either adds the current word to the dictionary (int standard mode), or cycles through the set of possible word fragments based on the current set of keystrokes for a given word (in predictive mode).

As things stand, the interface, and the standard mode have currently been completely implemented and your task is to complete the implementation of the predictive text mode. Specifically, you must finish off the implementation of the Dictionary component, which is used to store and retrieve words that correspond to sequences of keystrokes.

In terms of the implementation, it is expected that you will use a tree data structure. In this data structure, nodes represent both a keystroke (e.g. 2-9) and a list of word fragments are associated with the set of keystrokes that correspond to the path between the root node and the given node.

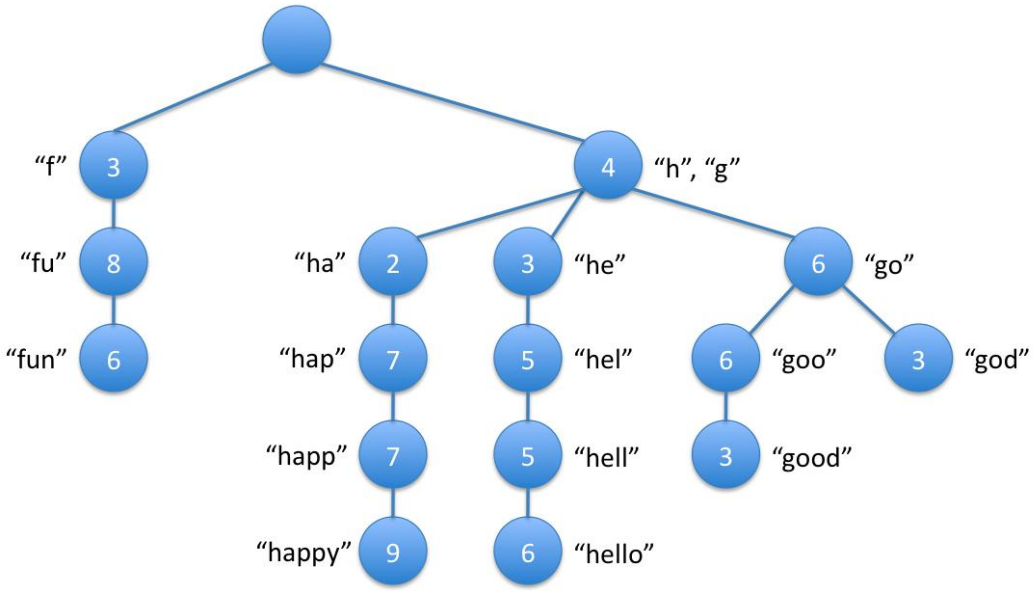

Figure 2: Sample Dictionary with five words encoded

An example tree is given above in figure 2. This tree encodes five words: "happy", "hello", "good", "god", and "fun". Based on this dictionary, pressing the following set of keys: 3,8,6 would result in the word "fun" being selected. In standard mode, this sequence of keystrokes would result in "dtm". To get "fun" in standard mode, you would have had to do the following: press 3 (3 times), press 8 (2 times), and press 6 (2 times).

Also, in predictive mode, pressing 4 would result in "h" being displayed, however, pressing "*", would cycle through the predictive options (i.e. pressing * once would result in "g" being displayed, pressing it a second time would result in "h" being displayed – "i" would never be displayed because no word beginning with this character is stored in the dictionary).

If no word exists the corresponds to the current sequence of keystrokes, then the predictive text system reverts to the standard text system and generates a word based on the first character associated with the keystrokes (i.e. if "fun" was not in the dictionary, then 3,8,6 would correspond to "dtm".

Part A: Basic Dictionary Implementation — Worth 60%

The first step in your assignent is to complete the implementation of the Dictionary component. This component makes use of two core classes: the main `Dictionary` class, and an inner class entitled `Keystroke`. The idea is that the inner class represents the data that is stored at each node in the tree, which is modeled using a standard (prewritten) implemenation of the Tree ADT. This implementation is a general tree implementation that allows each node to have any number of children.

Each question in this section is worth 15% of the final mark.

1) Start by completing the `Keystroke` class, which contains two fields: an integer value, key that represents the keystroke, and a list of Strings, which should hold all the word fragments that correspond to that keystroke. You must start by implementing the `addWord(...)` method.

   This method adds a word to the list of Strings only if the word is not already in that list (i.e. you need to search the list to see if the word you are adding is already in the list, and if it isn't then you should add it).

   Test your solution by creating a Keystroke object and adding four words to it (one being a repeated word) and print out the contents of the list (which is accessible via the getWord() method).

2) Next, you need to implement the `insert(...)` method that is part of the `Dictionary` class. This method takes a word (String) as input, and adds that word to the dictionary tree. To do this, you need to loop through the sequence of characters that make up the word, and map these characters onto the equivalent sequence of keystrokes (e.g. "great" would map onto the sequence 4,7,3,2,8).

   To help you implement this method, a static field of type Map has been created which is called characterMap. This map has been initialised so that each valid character ('a'-'z') is associated with the corresponding keystroke. The objective is that, you should traverse through the tree as you loop through the characters that make up the word. If no corresponding keystoke node exists, then you should create that node. For each keystroke, you should add the corresponding word fragment to the list of words associated with that keystroke. You should repeat this until the word is inserted into the dictionary.

   Test this by writing a test program that creates a Dictionary object and then add a sequence of words, after each insertion, print out the state of the Dictionary object (this is already provided) to see the current state of the underlying tree. Repeat this until you are happy that your implentation is correct.

3) Next, you must implement the `findWord(...)` method. This method takes a List of Integers (the sequence of keystrokes) as input, which is the internal representation (for the predictive text mode) of the word this is currently being typed.

The idea behind this method is to locate the current list of possible word fragments (if they exist) in the dictionary that corresponds to the sequence of keystrokes. To do this, you must use the sequence of keystrokes to traverse the tree to find the node that corresponds to that sequence. Once you identify that node, you can then use the getWord() method to return a reference to the list of possible words.

4) Finally, you must write a `load(...)` method, which takes a String as input, and which loads the file that is referred to by that String. The format of this file is a text file that contains one word per line. Some sample code for reading from such a file is:

```
BufferedReader in = null;
try {
        in = new BufferedReader(new FileReader(filename));
} catch (FileNotFoundException ex) {
        System.out.println("No Such File: " + filename);
}

try {
        String line = in.readLine();    // reads a line from the file or returns
                                        // null if there is no more to read
} catch (IOException ioe) {
        System.out.println("Error reading from: " + filename);
}

try {
        in.close();       // closes the file
} catch (IOException ioe) {
        System.out.println("Error closing: " + filename);
}
```

You should try to adapt this code to read all the data in a file. Test your implementation by creating and loading a sample dictionary file. Show the state of the dictionary after the file is loaded.

When you know that the system works, modify the load statement in the Main class to load the dictionary provided.

Part B: Advanced Problems — Worth 40%

These final questions relate to more advanced problems that you can try to solve to get additional marks. Each question in this section is worth 20% of the final mark.

1) Modify the implementation of the Keystroke class so that the List of Strings returned by the getWords() method is ordered based on the number of words in the dictionary that correspond to that word fragent. For example, if the dictionary contained the words “age” (243), “big” (244), and “bid” (243), then when the sequence of keystrokes entered is 24, the word fragment “bi” would lbe displayed (because two words matching this sequence are associated with this word fragment, while only one is associated with the word fragment “ag”).

2) Personalise the predictive text system so that words that the most frequently used word fragment for a given sequence of keystrokes is presented to the user. For example, in question B1, the sequence of keystrokes 24 would result in “bi” being displayed. However, if the user tended to use the word “age” rather than “big” or “bid”, then the predictive text system should (at some point) return the string “ag” for the sequence of keystroke 24.