About Lab 8

The first step of Lab 8 is to implement HashMaps. You will do a chained map, meaning that each entry of your hash array is a list of all of the <key, value> pairs where the key hashes to that index of the array.

The MyHashMap<K, V> class has a number of methods that should be easy to create and easy to test:

int size()
boolean isEmpty()
void clear()
String toString()
V get(K key)
V put(K key, V value)
V remove(K key)
boolean containsKey(K key)
boolean containsValue(V value)

There are also two methods that might make you think a bit:

Iterator<K> keys()

Iterator<V> values()

Finally, there is a resize() method. For this you need to build a new hash array and rehash everything to it.

The other part of the lab concerns Markov models, which we'll use for language generation. The basic idea is easy. Suppose you have a sample of text on which we are basing our generated text. Let's consider substrings of length k=2. In the sample, the substring "bo" is followed 50% of the time by the letter 'b', and 50% by 'm'. As we are generating text, if we output the letters 'b' and 'o' (so we have the substring "bo", we randomly choose one of the followups 'b' and 'm', with probability 0.5.

Say we generate 'm'. Now our most recent string of length k is "~~b~~o"+"m" = "om". Suppose every time "om" appears in the sample it is followed by 'b'. We do the same, and so we have now produced "bomb". This continues as long as we wish.

There are two things to note here:

a) We aren't considering semantics at all. There is no "meaning" to the generated text; it is just a sequence of characters.
b) If you keep k small (such as the value 2 in our examples), the resulting text is rather random and meaningless. However, if you make k fairly large, on the order of 10 or so, the generated text will consist mostly of real words.

People who are serious about generating language usually start with a semantic model and a model of sentence structure, and then use Markov models (perhaps on words rather than individual characters) to add some variation and individual style to the generated sentences.

We will make a class Markov to represent each substring of length k. The class holds the substring and a TreeMap<Character, Integer> that keeps track of all of the followup characters. Our Markov model is a hashmap of all of these Markov instances: HashMap<String, Markov>.

When we are building up our model, each time we read a letter x we look at the previous string of length k. We use this string to lookup the corresponding Markov object in our HashMap. We add x as a character following the substring in its Markov object. We then update the substring by dropping its first letter and adding x onto its end.

For generating text, we start with the first k letters of our text sample. Now, suppose variable sub holds the most recent substring of k letters. We get the Markov object associated with sub in the HashMap. Suppose the Markov object tells us that sub was followed in the sample 2 times by the letter 'a', 5 times by 'b' and 3 times by 'c' (for a total of 10 entries). Choose a random number between 0 and 9. If this is less than 2, choose the letter 'a'; if it is at least 2 but less than 7 choose 'b', and if it is 8 or greater choose c. Output this letter, update sub (by dropping its first letter and adding the one just chosen), and repeat.

There is one more issue. It is actually rather awkward to use our usual Scanner to read the text sample -- Scanner wants to read words (tokens) or entire lines, and our algorithm wants one letter at a time (counting end-of-line markers and other white space as letters).

The following demo program uses class FileReader to echo an input file one letter at a time.

```
import java.io.*;
public class Test {
        public static void main(String[] args) {
        int nextChar;
        FileReader input=null;
        String inputFileName = args[0];
        try {
             input = new FileReader(inputFileName);
        } catch (FileNotFoundException e) {
              System.err.println("Could not open file "+inputFileName+": "+e.getMessage());
              System.exit(2);
        }
        try {
              while ( -1 != ( nextChar = input.read() ) )  {
                    char c = (char) nextChar;
                    System.out.print(c);
               }
        } catch (IOException e) {
              System.err.println("Error reading from"+inputFileName+": "+e.getMessage());
               System.exit(4);
        }
   }
}
```

If you prefer, the second try-block could also be written

```
try {
        boolean done = false;
         while (!done) {
                 nextChar = input.read();
                if (nextChar == -1)
                         done = true;
                else {
                        char c = (char) nextChar;
                        System.out.print(c);
                }
          }
}
```