CSCI 2110 Computer Science III
Data Structures and Algorithms
ASSIGNMENT NO. 3

Date given: Friday, October 27, 2017
Due: Wednesday, November 15, 2017 (11.55 p.m.)

The objective of this assignment is to implement the Huffman coding algorithm using the binary tree data structure.

Problem Summary: Read a text file. Count the number of occurrences of each symbol (character) in the file. You can ignore the spaces and the newline returns, but count the alphabet, digits, and punctuations. Convert the frequencies into probabilities. Build a Huffman tree with the symbols and their probabilities. Derive the Huffman codes. Encode the text file with the codes. Decode the encoded text file and show that it is the same as the input text file.

Download the BinaryTree.java next to the assignment link. You may add extra methods or make appropriate changes to the methods if necessary. You may also add other classes.

Problem in Detail:

In order to help you with the assignment, here's the Huffman algorithm step-by-step procedure (as discussed in the lectures). You are not obligated to follow this procedure. Any valid method of deriving Huffman codes is acceptable, but it must use the binary tree data structure.

Step 1: Read a text file and parse it into individual characters. Count the number of characters of each type to get the frequency of each character, that is, the number of times each character appears in the text file. You can ignore the spaces and the newline returns, but count the alphabet, numbers, and punctuations.

Here's a sample text file that you can use to test your program. This is the same file that will be used to mark your assignment. You can download it from the web page next to the assignment link.

```
POKEMON TOWER DEFENSE
YOUR MISSION IN THIS FUN STRATEGY TOWER DEFENSE GAME IS TO
HELP PROFESSOR OAK TO STOP ATTACKS OF WILD RATTATA. SET OUT ON
YOUR OWN POKEMON JOURNEY, TO CATCH AND TRAIN ALL POKEMON AND
TRY TO SOLVE THE MYSTERY BEHIND THESE ATTACKS. YOU MUST PLACE
POKEMON CHARACTERS STRATEGICALLY ON THE BATTLEFIELD SO THAT
THEY STOP ALL WAVES OF ENEMY ATTACKER!!!
DURING THE BATTLE YOU WILL LEVEL UP AND EVOLVE YOUR POKEMON.
YOU CAN ALSO CAPTURE OTHER POKEMON DURING THE BATTLE AND ADD
THEM TO YOUR TEAM. USE YOUR MOUSE TO PLAY THE GAME.
GOOD LUCK!
```

The outcome of the first step is a list of symbols and their frequencies. For example,
P 14
O 49
K 11
E 53
etc.

Step 2: Convert the frequencies into probabilities. For this, divide each frequency by total number of characters. For example, the above text file has 455 characters without spaces. You would divide each frequency by 455. The outcome of this step is a list of symbols and their probabilities. For example,

Symbol	Probability
P	0.031
O	0.108
K	0.024
E	0.116

etc.

Note that if this step is done correctly, the sum of all the probabilities is 1.0. Sometimes, you may have to round it up to the nearest decimal value, and that approximation is OK. In the above example, the probabilities have been rounded to three decimal places. You can use a different precision but make sure that they all add up to 1.0 (or very close to it).

To do steps 1 and 2, you can create a class called Pair.java that defines the symbol and its probability as an object.

```
public class Pair
{
   private char value;
   private double prob;

   //constructor
   //get and set methods
   //toString method
}
```

You can create an arraylist of Pair objects and store the items into the arraylist as you read them. Of course, you will need other variables and methods to count the frequencies and convert them into probabilities.

Step 3: Using this set of symbols and frequencies, build the Huffman tree.

Step 3.1: Create a queue of Binary Tree nodes. Each Binary Tree node is of type Pair. The queue can be implemented as a simple arraylist, where enqueue means adding an item to the end of the arraylist and dequeue means removing the item at index 0. That is, the queue is an arraylist of type <BinaryTree<Pair>>. The queue contains these sorted according to the increasing order of their frequencies. This is your Queue S. This is done by checking the

arraylist for values in increasing order, creating the binary tree nodes and enqueueing them in the queue.

If you enumerate the Queue S, it should have the Pair objects in increasing order of their frequencies, something like this:

('!', 0.009), ('V', 0.011), etc.

Step 3.2 : Now initialize another queue T (another arraylist) of type <BinaryTree<Pair>>.

Step 3.3 : Build the Huffman tree according to the algorithm discussed in the lectures.

For instance, in the above example, first ('!', 0.009) and ('V', 0.011) will be dequeued from S. Create a node with the combined frequency. What do you put as the character for the combined node? You can put a dummy character, say '0'. So ('0',0.02) will be the parent node, and ('!', 0.009) and ('V', 0.011) will be the left and right children. This tree will be enqueued to Queue T.

You keep repeating the above procedure and building the Huffman tree according to the algorithm given below:

Pick the two smallest weight trees, say A and B, from S and T, as follows:

a) If T is empty, A and B are respectively the front and next to front entries of S. Dequeue them from S.

b) If T is not empty,
i) Find the smaller weight tree of the trees in front of S and in front of T. This is A. Dequeue it.
ii) Find the smaller weight tree of the trees in front of S and in front of T. This is B. Dequeue it.

3. Construct a new tree P by creating a root and attaching A and B as the subtrees of this root. The weight of the root is the combined weights of the roots of A and B.

4. Enqueue P to T.

5. Repeat steps 2 to 4 until S is empty.

6. After step 5, if T's size is > 1, dequeue two nodes at a time, combine them and enqueue the combined tree until T's size is 1. The last node remaining in the queue T will be the final Huffman tree.

Step 4: Derive the Huffman codes.

The following methods can be used for finding the encoding. They use a String array of 256 to cover all characters in the Unicode set. You can tweak them if necessary.

```
public static void findEncoding(BinaryTree<Pair> t, String[] a, String prefix)
{
   if (t.getLeft()==null&& t.getRight()==null)
   {
      a[(byte)(t.getData().getValue())]= prefix;
```

```
  }
  else
  {
    findEncoding(t.getLeft(), a, prefix+"0");
    findEncoding(t.getRight(), a, prefix+"1");
  }

}

public static String[] findEncoding(BinaryTree<Pair> t)
{
  String[] result = new String[256];
  findEncoding(t, result, "");
  return result;
}
```

Put the resulting Huffman codes into an output text file, call it Huffman.txt. The result would look something like this:

Symbol	Prob.	Huffman code
!	0.009	010000
V	0.011	010001

etc.

The codes are fictitious values, but you get the idea.

Step 5: Read the sample Pokemon text file (call it Pokemon.txt) and Huffman.txt file, and encode it using the Huffman symbols. Do not encode spaces and newline characters. Leave them as they are. Write the encoded file into another text file, Encoded.txt.

Step 6: Read the encoded text file and decode it. Write the decoded file into yet another text file, Decoded.txt.

If you have done everything correctly, then Decoded.txt must be the same as Pokemon.txt.

Submit a zip file containing all the source codes (.java files), Pokemon.txt, Huffman.txt, Encoded.txt and Decoded.txt.