CSCI 2110 Data Structures and Algorithms

Fall 2023

Assignment No. 5
Date Given: Monday, November 13th, 2023
Due (on Brightspace): Monday, November 27th, 2023, 11.59 PM

This assignment in on the Huffman coding algorithm using the Binary Tree Data Structure that we discussed in the lectures. Please review Module 5: Binary Trees before you start the assignment.

You will need the following files to complete this assignment. Download them.

BinaryTree.java (Generic Binary Tree Class)
LettersProbability.txt (Text file containing letters and their probabilities)

<u>Problem Summary:</u>

Your program should do the following:

1. Read the LettersProbability.txt file.
2. Build the Huffman tree and derive the Huffman code for each symbol (letter).
3. Prompt the user to enter a line of text. Read the line and encode it as a String of 1's and 0's using the Huffman codes.
4. Decode the String of 1's and 0's and display the decoded text. If your program is correct, the decoded line that is displayed will be identical to the line of text entered by the user.

Heres' a sample dialog:

```
Huffman Coding
Enter the name of the file with letters and probability: LettersProbability.txt

Building the Huffman tree ….
Huffman coding completed.

Enter a line (uppercase letters only): THIS IS COOL
Here's the encoded line:  0100010010010100000 0001011001 10101010000100001101010O
The decoded line is:  THIS IS COOL
```

Note 1: **<u>The above codes are not the actual codes that you will derive. They are just for demonstration only. Your codes will be different.</u>**

Note 2: You will be encoding only uppercase letters. Keep the spaces as they are in the input. The spaces are not encoded.

<u>Problem in Detail:</u>

You will write a class that builds the Huffman tree given a list of symbols-probability pairs and provides methods to encode and decode text according to the Huffman coding algorithm. Call your class file **Huffman.java**. You can design as many methods as you find appropriate within the program to make it modular.

Step 1: Read the LettersProbability.txt file. The file contains the uppercase letters of the English alphabet and their probabilities arranged in increasing order of probability. For example, the first few lines of the text file are as follows:

```
Z     0.0007
J     0.0010
Q     0.0011
X     0.0017
```

etc.

If you add up all the probabilities, you will see that they add up to 1.

Step 2: You will need to keep track of letters and their relative probabilities in order to build your Huffman tree. To do this, you will likely find it useful to create a separate class called Pair.java:

```
public class Pair implements Comparable<Pair>{
  // declare all
  //required fields
  private char value;
  private double prob;

  //constructor
  //getters
  //setters
  //toString

  /*
  The compareTo method overrides the compareTo method
  of the Comparable interface.
  */
  @Override
  public int compareTo(Pair p){
    return Double.compare(this.getProb(), p.getProb());
  }
}
```

As you read each line of the LettersProbability.txt file, you will create a **BinaryTree<Pair>** object for each letter-probability pair that you read.

Step 3: Next, you will build a Huffman tree. To build a Huffman tree you will require two queues, Queue S and Queue T, of type **BinaryTree<Pair>**. You may use structures from the Java Standard Library, or you may choose to trivially implement your queues using **ArrayLists** (appending new elements and removing at index 0). Both options are acceptable.

- Enqueue the BinaryTree<Pair> objects in ascending order into the Queue S, with the lowest probabilities at the head of the queue. Your second queue (Queue T) should remain empty for the time being.

- Now you can implement the rest of the Huffman algorithm from your workbook:

  1) Pick the two smallest weight trees, say A and B, from queues 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.

  2) 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. (You may find it useful to assign a character value to P that is not an uppercase letter. A 0 (zero) character will work.

  3) Enqueue tree P to queue T.

  4) Repeat the previous steps until queue S is empty.

  5) If the number of elements in queue T is greater than 1, dequeue two nodes at a time, combine them (see strep 2) and enqueue the combined tree until queue T's size is 1. The last node remaining in the queue T will be the final Huffman tree.

Step 4: You're now ready to derive the Huffman codes. The following methods can be used to find the encoding:

```
private static String[]
  findEncoding(BinaryTree<Pair> bt){ String[] result
  = new String[26];
  findEncoding(bt, result,
  ""); return result;
}

private static void findEncoding(BinaryTree<Pair> bt, String[] a, String prefix){
  // test is node/tree is a leaf
  if (bt.getLeft()==null && bt.getRight()==null){
    a[bt.getData().getValue() - 65] = prefix;
  }
  // recursive calls
  else{
    findEncoding(bt.getLeft(), a, prefix+"0");
    findEncoding(bt.getRight(), a, prefix+"1");
  }
}
```

Step 5: Now that you have the Huffman codes, you are ready to encode and decode text. Prompt the user to enter a line of text in uppercase letters.

```
Enter a line of text (uppercase letters only): THIS IS COOL
```

Display the encoded line

```
Here's the encoded line:   0100010010010100000  0001011001 1010101000010000011010100
```

Next decode the above line and display the original line.

```
The decoded line is:           THIS IS COOL
```

Note that you are leaving the spaces as they are. Further, the binary numbers that are displayed are just a String of characters of 1s and 0s.
You can use an ArrayList of characters to store the input line and the encoded line (one ArrayList for each).

If your program is correct, the decoded line will be identical to the line that was entered by the user.
You are done!
Again, **The above codes are not the actual codes that you will derive. They are just for demonstration only. Your codes will be different.**

What to submit: ONE zip file containing
1. Your Huffman.java file, Pair.java file and any other helper class files that you created.
2. BinaryTree.java file that was given to you.
3. LettersProbability.txt file that was given to you.
2. A text or PDF containing screenshots of at least three different runs of the program showing the encoding and decoding.

You MUST SUBMIT .java files that are readable by your TAs. If you submit files that are unreadable such as .class, you will lose points. Please additionally comment out package specifiers.

The TAs will test your program against their own input lines.

Late Submission Penalty: The assignment is due on Monday at 11.59 PM. Late submissions up to 5 hours (4.59 AM on Tuesday) will be accepted without penalty. After that, there will be a 10% late penalty per day on the mark obtained. For example, if you submit the assignment on Tuesday at 12 noon and your score is 8/10, it will be reduced to 7.2/10. Submissions past five days after the grace submission time will not be accepted.