TCSS 342A - Data Structures

Group Project 2 - Compressed Literature

Due Date: February 21, 2017 (Tuesday) 9:30am via Canvas

This group project consists of programming and written work. Solutions should be a complete working Java program including your original work or **cited contributions** from other sources. These files should be compressed in a .zip file for submission through the Canvas link.

This project is to be completed on your own or in a group of 2 or 3 members. One submission per group. State the official names (as appear in myUW) of all your group members clearly in your submission.

This project will be graded out of a total of 16 points.
Breakdown of points: 10 (code compiles and pass tests) + 6 (documentation) + 2 (bonus)

Project Description

Standard encoding schemes like ASCII are convenient and relatively efficient. However we often need to use data compression methods to store data as efficiently as possible. I have a large collection of raw text files of famous literature, including Leo Tolstoy's War and Peace consisting of over 3 million characters, and I'd like to store these works more efficiently. David Huffman developed a very efficient method for compressing data based on character frequency in a message.

In this assignment you will implement Huffman's coding algorithm in a CodingTree class. This class will carry out all stages of Huffman's encoding algorithm:

- Counting the frequency of characters in a text file.
- Creating one tree for each character with a non-zero count.
  - The tree has one node in it and a weight equal to the character's count.
- Repeating the following step until there is only a single tree:
  - Merge the two trees with minimum weight into a single tree with weight equal to the sum of the two tree weights by creating a new root and adding the two trees as left and right subtrees.
- Labeling the single tree's left branches with a 0 and right branches with a 1 and reading the code for the characters stored in leaf nodes from the path from root to leaf.
- Using the code for each character to create a compressed encoding of the message.

You are also responsible for implementing a Main controller that uses the CodingTree class to compress a file. The main must:

- Read the contents of a text file into a String.
- Pass the String into the CodingTree in order to initiate Huffman's encoding procedure and generate a map of codes.
- Output the codes to a text file.
- Output the compressed message to a binary file.
- Display the size of the compressed text, compression and run time statistics.

Formal Specifications

You are responsible for implementing the CodingTree class that must function according to the following interface:

- void CodingTree(String message) - a constructor that takes the text of a message to be compressed. The constructor is responsible for calling all private methods that carry out

the Huffman coding algorithm.
- Map<Character, String>codes - a **public** data member that is a map of characters in the message to binary codes (Strings of '1's and '0's) created by your tree.
- String or List<Byte> bits - a **public** data member that is the message encoded using the Huffman codes.

To implement your CodingTree all other design choices are left to you. It is strongly encouraged that you use additional classes and methods and try to use the proper built in data structures whenever possible. For example, in my sample solution I make use of a private class to count the frequency of each character, a private node class to implement my tree, a recursive function to read the codes out of the finished tree, and a priority queue to handle selecting the minimum weight tree.

You will also create a Main class that is capable of compressing a number of files and includes methods used to test components of your program.
- void main(String[ ] args) - this method will carry out compression of a file using the CodingTree class.
  - Read in from a text file. You may hardcode the filename into your program but make sure you test with more than one file.
  - Output to two files. Again feel free to hardcode the names of these files into your program. These are the codes text file and the compressed binary file.
  - Display statistics. **You must output the original size (in bits or bytes), the compressed size (in bits or bytes), the compression ratio (as a percentage) and the elapsed time for compression.** The "elapsed time" could include the reading of the input and writing of the output files.
- Test files - include at least one test file that is a piece of literature of considerable size.
- Check out Project Gutenberg (https://www.gutenberg.org/) an online database of literature in text format.

(Bonus) Implement your own MyPriorityQueue<T> class using the array implementation mentioned in lecture. Use it in place of the Java PriorityQueue in your CodingTree class.

Documentation
In addition to this programming assignment, you will include **documentation of how you tested your code**. Submit your test input files, and explain what conditions that your test input files test for. See programming guidelines on Canvas for additional guidance. **For each of your test input files, clearly state the size of the original text (in bits) and the size of the compressed text (in bits) in "doc.pdf".**

Submission
The following files are provided for you:
- WarAndPeace.txt - the plain text of Leo Tolstoy's novel War and Peace.
- codes.txt - An appropriate set of codes produced by my sample solution. (Note: This is not a unique solution. Other proper encodings do exist.)
- compressed.txt - The compressed text of the novel. Its size is the most relevant feature.

You will submit a .zip file containing:
- Main.java - the simulation controller.
- CodingTree.java - the completed and functional data structure.
- <sample>.txt - a novel or work of art in pure text for that you have tested your program

on. (you can submit more than one input test files)

- compressed.txt - the compressed version of your selected text. (you can submit more than one output files) I will accept compressed output in the format of the provided sample "compressed.txt" or binary 0/1's.
- codes.txt - the codes produced on your selected text.
- **(Bonus)** MyPriorityQueue.java - an array based implementation of PriorityQueue.
- Doc.pdf - a pdf explaining how you tested your code, and the size of the compressed text for each of your input test files. If you implemented the bonus option, please indicate in this file.

Points will be deducted if you do not follow the above file format or filenames.

Grading Rubric
The following is a detailed description of what I am looking for when grading your project.

CodingTree

- Counts the characters in the message and stores the count in an appropriate data structure.
- Initialize a single tree for each character.
- Build the Huffman tree using and efficient data structure.
- Recursively extract the codes from the Huffman tree.
- Encode the text.

Main

- Read text from input file. You can assume that you compress one file at a time.
- Output codes to text file.
- Output encoded text to binary file.
- Component testing or debugging code.
- Display run statistics.
- Extra test files.

MyPriorityQueue **(***Bonus***)**

- All methods used are implemented.
- All methods are correct.
- All methods are efficient.

Documentation

- Documentation of how you tested your code.
- For each input test files you submit, state clearly the size of the compressed text (in bits or bytes).

Tips for maximizing your grade:

- Make sure your classes match the interface structure exactly. I will use my own controller (Main.java) and test files on your code and it should work without changes. Do not change the method name (even capitalization), return type, argument number, type, and order. Make sure all parts of the interface are implemented.
- Submit the .java files. If you use eclipse these are found in the "src" directory, not the "bin" directory.
- I will grade your project using Java 8.

- All program components should be in the default package. If you use a different package it increases the difficulty of grading and thus affects your grade.
- Place your name in the comments at the top of every file. If you are a group of two, make sure both names appear clearly in these comments.

Grading logistics
I will compile your code using
```
javac *.java
```

I will use my own “Main.java” to test your code. You are free to write your own Main.java, but your routines must work with my “Main.java”.

Sample input file: (`WarAndPeace.txt`)

I will execute your code
```
java Main WarAndPeace.txt
```

Expected output:
```
Uncompressed file size: 3291623 bytes
Compressed file size: 1875127 bytes
Compression ratio: 56%
Running Time: 3543 milliseconds
```