# TCSS 342 - Data Structures
# Assignment 3 - Compressed Literature

*Due Date: Wednesday, November 13th*

## Guidelines

This assignment consists of programming work. Solutions should be a complete working Java program including both your work and any files provided by the instructor used in your solution. These files should be compressed in a zip file for submission through the moodle link.

This assignment is to be completed on your own or in a group of two. If you choose to work in a group of two this must be clear in your submission. Please see the course syllabus or the course instructor for clarification on what is acceptable and unacceptable academic behavior regarding collaboration outside a group of two.

## Assignment

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 Ayn Rand's Atlas Shrugged 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 by:

- counting the frequency of characters in a text file.
- creating a tree with a single node for each character with a non-zero 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.
- labelling 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.

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

- void CodingTree(List<Character> chars) - a constructor that takes the text of a message to be compressed. The constructor is responsible for calling all methods that carry out the Huffman coding algorithm and ensuring that the following properties have the correct values.
- String codeStr - a String property that has one Code per line, each code consisting of a character and a binary codeword.

- List<Code> codes - a list of Code objects, one for each character in the text to be encoded.

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 built in data structure 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.

The following files are provided for you:
- Main.java - a controller that instantiates the CodingTree and carries out the actual encoding.
- Code.java - a simple data wrapper consisting of a character and a codeword.
- Rand, Ayn - Atlas Shrugged.txt - the plain text of Ayn Rand's novel Atlas Shrugged. (Note: It is against copyright law for you to read this novel unless you own it. Compression only please!)
- 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. It's size is the most relevant feature. (Note: Please read this version freely.)

You will submit a .zip file containing:
- Main.java - the simulation controller as provided or with only minor modifications accepted.
- CodingTree.java - the completed and functional data structure.
- Code.java - in case you have modified it to suit your solution.
- <sample>.txt - a novel or work of art in pure text for that you have tested your program on.
- compressed.txt - the compressed version of your selected text.
- codes.txt - the codes produced on your selected text.

Grading:
- 80% Correctness - your solution must implement Huffman's algorithm as closely as possible and produce codes that provide compression equal to my sample solution.
  - Part marks available for partial correctness based on code inspection.
- 10% Organization and Style - I judge your design choices regarding naming conventions, modularization, use of whitespace, indentation.
- 10% Documentation - sufficient comments should be added to major methods, complicated blocks of code, or variables with unclear intended uses.

Extra Credit:
- For 10% extra credit implement a decoder that will take the two files codes.txt and compressed.txt and produce the decoded manuscript.