# CptS 233 PA #2

In this task, you will implement a program that uses Huffman Codes (book p. 435) to compress files. Like the micro assignments, I provide quite a bit of starter code. To complete this assignment, you must use the following data structures:

* ArrayList for bit storage, text storage, and Huffman tree storage
  + Note that a better implementation would use priority\_queue for tree storage. But we are trying to keep it focused upon the implementation, so don’t use priority queue.
* HashMap for storage of Huffman Maps
* Custom HuffmanNode, HuffmanInternalNode, HuffmanLeafNode, and HuffmanTree classes to represent a Huffman Tree.

In completion of this assignment, you must implement the following functions in PA2.java:

### findSmallestTree

Finds the smallest tree (by weight) in the supplied forest. Note that this function accepts a second optional parameter of an index to skip. Use this index to allow your function to also find the 2nd smallest tree in the forest.

### huffmanTreeFromText

Builds a Huffman Tree from the supplied vector of strings. This function implement's Huffman's Algorithm as specified in page 435 of the book. **IMPORTANT!** In order for your tree to be the same as mine, you must take care to do the following:

1. When merging the two smallest subtrees, make sure to place the smallest tree on the left side!
2. Have the newly created tree take the spot of the smallest tree in the forest (e.g. List.set(smallest\_index, merged\_tree) ).
3. Use List.remove(second\_smallest\_index) to remove the other tree from the forest.

### huffmanTreeFromMap

Generates a Huffman Tree based on the supplied Huffman Map. Recall that a Huffman Map contains a series of codes (e.g. 'a' => 001). Each digit (0,1) in a given code corresponds to a left branch for 0 and right branch for 1. I used recursion to solve this problem.

### huffmanEncodingMapFromTree

Generates a Huffman Map based on the supplied Huffman Tree. Again, recall that a Huffman Map contains a series of codes (e.g. 'a' => 001). Each digit (0,1) in a given code corresponds to a left branch for 0 and right branch for 1. As such, a given code represents a pre-order traversal of that bit of the tree. I used recursion to solve this problem.

### writeEncodingMapToFile

Writes the supplied encoding map to a file. My map file has one association per line (e.g. 'a' and 001). Each association is separated by a sentinel value. In my case, I went with a double pipe (||).

### readEncodingMapFromFile

Creates a Huffman Map from the supplied file. Essentially, this is the inverse of writeEncodingMapToFile. Be sure to use String.split() function to make your life easier! Note that String.split() accept a parameter of regular expression, not just a regular string. So, if you are splitting based on the sentinel you defined in writeEncodingMapToFile(), and you are using double pipe as your sentinel, you should write it as: String.split(“\\|\\|”)

### decodeBits

Uses the supplied Huffman Map to convert the vector of bits (bools) back into text. To solve this problem, I converted the Huffman Map into a Huffman Tree and used tree traversals to convert the bits back into text.

### toBinary

Uses the supplied Huffman Map to convert the supplied text into a vector of bits (bools).

## Deliverables

You must commit your program to your github repository no later than due date (TBD), and submit your commit hash to the Blackboard dropbox.

**NOTE:** for this assignment, you do not need to worry about encoding/decoding line breaks in the original files. If you figure out how to, that would earn you some extra credits.

## Grading Criteria

This is a quite challenging assignment. Further assistance and instruction will be given out periodically.

Your assignment will be judged by the following criteria:

* [80] Test cases (20pts / ea). For each test case, your program should compress and/or decompress a given file correctly. You may compare your results with mine in **PA2-ref.zip** – they should be identical (regardless of the line break) to be correct. The output files in the PA2-ref.zip will be explained in class.
* [10] Data structure usage. Your program only uses Huffman Trees, HashMaps, and ArrayLIsts. Base arrays, linked lists, priority queues, and any other custom container are not used.
* [10] Your code is well documented and generally easy to read.