Adaptive Huffman Encoding Documentation

Overview And Algorithm Logic: The program encodes and decodes text file using adaptive Huffman encoding. **Logic used for encoding :** First the characters in the file are stored in a single character array then each character is taken and if its a new character then its added to the intermediate tree or else its frequency is increased, this is continued until the level is reached once the level is reached then this intermediate tree is rebuilt using basic Huffman encoding logic. I am copying the data from a sorted array to a queue and removing top two elements of the queue and combining them to form new node, then all this data is put into an array from a queue and sorted, then again added back to the queue until queue has only one element left. Follow this until all the characters are finished.

Logic for decoding: the characters are stored in a character array, If its not a leaf node then traverse down the tree based on 1's and 0's until leaf node is reached, once its reached if it's a new character node then add the character to decoded string or else increase the frequency. And once the level is reached then rebuilt logic of same encoding is used.

Files and relevant data; the submitted files contain TextCompressImplementation class which implements FileCompressor interface, contains the methods for encoding, decoding and codebook an all the remaining logic. TO EXECUTE, AN OBJECT OF "TextCompressImplementation" is to be created and encode and decode and codebook functions are to be called using that object. Other files include Node class for each node of tree and Queue class for Queue data structure implementation.

Data Structures and ADT's used: I'm using a HashMap to store my character value and its encoded value for the ease of finding the value of a given key. A treeMap to sort the elements of map based on their ascii value. An array of Nodes that contain information of its left and right child for ease of sorting it based on the frequency of each node. A queue for the ease of removing the two smallest element from it while rebuilding the tree using Huffman encoding. Due to constraint of not being able to use data structures from collection framework and time constraint I couldn't use Priority Queue. If i was allowed to use priority queue then I wouldn't have used treeMap, array and queue to sort elements based on frequency and remove first two elements and adding them to form Huffman encoding logic. Hence by using priority queue my code could be more efficient.

Assumptions:

- If the text is finished before the level of reset is reached then the tree is rebuilt and the codebook contains encoded values of characters after the rebuilt tree.
- If level is 0 then output of encoding contains Bits for END-Of-File after the tree is rebuilt

- Text file is of fixed size and fixed data i.e data in file is not increased while program is running
- If Corrupted file, then program exits printing “Corrupted file”
- If empty file is passed program prints “Empty file” and exits

Limitations: The current implementation would fail to work if the file is of increasing data i.e the data to file is added while program is running.

Refrences:

- To familiarise myself with treeMap syntax this is the website I used (**https:// www.baeldung.com/java-treemap**).
- To learn how to create a file using Java this thread on StackOverFlow was referred (**https://www.baeldung.com/java-treemap**)