Heaps: Huffman Encoding

Huffman Encoding is a popular algorithm used in text compression. The technique revolves around the use of a Priority Queue (Heap) and Binary Tree in order to achieve an optimal compression for static text.

Overview

In this lab, you will implement the Huffman algorithm to compress (and decompress for bonus) text files. Your program should be able to read in a text file chosen by the user and compress it to an appropriate file which the user should also choose.

Concepts

- Generics with Requirements
- Priority Queues
- Binary Trees
- File Input and Output

Program Details

- The program must be entirely console based and accept command-line arguments. -o <outputfile> indicate the file to create. The last command-line argument is the file to decompress. For example, java program.class -o output.txt input.txt will read from the file input.txt and output to the file output.txt.
- There must be a clear separation of presentation (file I/O, console I/O, etc) and logic (Heap, Huffman Encoder/Decoder, Graphics, etc).
- The output file must contain a Text Representation of the Huffman Dictionary.

- The program must be able to handle compression up to the Huffman Dictionary.
- Java-doc style comments are required for **all** non-private method and variables. For private variables, commenting is required if the code is confusing or not self-documenting.

Bonus

Compress the file all the way to binary and then allow for decompressing the file.

- Store the tree in binary prior to the message.
- Add an End Token at the end of the message
- Add a -d command-line argument to indicate that you're decoding the input file. You can also attempt to auto-detect what to do based on the data in the file!

Grading

- /4 Separation of logic and presentation
- /10 Implementation of a Priority Queue
  - /3 Add method
  - /4 Remove method
  - /3 Peek, isEmpty
- /5 Implementation of the Huffman Encoding
- /3 Implementation of File Input and Output
- /3 Documentation and style (class and method documentation, name in program, etc.)
- Bonus: Storing and Decompressing Huffman text files
  - Reading and Writing Binary Files

    - Dealing with Binary Data
    - Decoding information correctly

Some Hints

1. Design your classes first! The scoring guide will help.
2. Deal with and **test** compression without file output first! Then deal with file output. Then move on to decompression (in the reverse order).
3. Remember that compressed data is BINARY and cannot be handled exactly like text. (Use BitSet, byte[], bit shift operators, etc). Research (read: Google) how to deal with binary data in Java.