Aaron Davis

COSC 310

12/11/2016

Huffman Encoder

Conclusion:

For this project, I began it with the idea of using a HashMap and <Key, Value> Pair stored within an ArrayList, from then, calculating the information necessary. Soon before this, I realized I needed my own custom Priority Queue class. With the Priority Queue, I would also need a Heap class. After looking further into making a Huffman tree and seeing it’s similarity to Binary trees, I also came to the conclusion I needed a custom Node class. After completing the Node class, I started on the Heap. The Heap and Priority Queue class actually have a nice implementation and explaining in the textbook. Since I didn’t quite need generics and other things the textbook offered, I used stack overflow to help with the implementation of both classes.

The Huffman code was the last worked on, while on stack overflow, I saw someone’s “run” method which called all the methods he needed in his tree class, then he simply calls that method when instantiating this. I decided that would be best for my implementation since I was already unfamiliar with building trees. The read file and print out methods were the easiest. In the readFile method Characters are represented as ints for the purpose of calculating frequency. My numOfCharacters method wasn’t necessary for this implementation, but with forward thinking, this would have calculated all unique characters and how frequently they appear. I found the HashMap to be not as easy to manipulate. fillQueue was the second method, which does exactly as a person would assume. The depthFirst meth and huffmanTree methods were my most difficult and last methods created. When trying to make these two methods, I realized I wasn’t building my tree correctly, luckily someone hand the same problem. Many people’s index’s started at 0, the Stack Overflow method began at 1, which I’m still unsure of. None the less, depthFirst was the last method to get solved. There were many different styles of implementations, but they all led to the same conclusion, recursion of left first then right. Everything in the else statement comes because the index of the code must be correctly placed. Realizing my indexes were off and I was missing some numbers, I realized it was better to re-do all my indexes so they’re all the same.