Lab 3 Analysis of Huffman Coding Design

EN. 605.202 Data Structures

By Eduardo Carrasco Jr

Date 4/15/25

Summary

Using the concepts of Huffman coding, I designed a Min Heap to read in a frequency table. Transcribe these results into a Min heap tree. I then use the min-heap tree to encode and decode text. Overall, the efficiency and choice of the min-heap tree is O(n*log n) time complexity. More details about the design choice can be found in this lab's Justification for Algorithm Design.

Enhancements

As python and software development have been incorporated widely in industry. It is standard to include two things to run a python module.

- Virtual Environment.
- Unit testing.
- Huffman node Class. Modular frequency table.

A unit testing file has been included in this module as well as the virtual environment instructions in the README.md.

As an extra enhancement the Huffman node can create a new custom frequency table in the parameter if a new string is used to encode create a new Huffman tree. See unit testing for options.

Time and Space Complexity

The first step is to read the frequency table file which is O(n*n) complexity.

Then convert the frequency table to a Huffman tree using Min Heap node structure. This choice was made due to the efficiency of the heap binary heap design of insert O(1), Find-min O(1) and delete of O(log n). Creating the min and sorting it will be in O(n * log n) time.

Final step is decoded and encode the binary numbers or clear text which takes O(1*n) n for the number of numbers/characters being decoded/encode.

Reflections and Learnings

The most challenging part was converting the frequency table to Huffman coding. I am still not sure if the answers are correct because it is not English words. I was expecting English phrases.

The second most challenging part was encoding the text from Huffman heap to binary values. Again I am not sure I implemented this correctly.

As the implementation for just English Alphabet, I can only image the complexity of using all ascii character would look like. The Heap would be a massive tree. Similar idea would be when you create a Huffman Coding for a language with more complicated languages such as Japanese or Chinese where a new character has a new meaning.

I learned a lot about recurve pythonic functions. I used a few in this assignment. Fun way to create a driver method to help with the recursion.

Justification for Algorithm Design

In this lab the goals was to read on a frequency table first. This was read in and converted to a tuple to then pass into a Huffman tree method. This method converts the frequencies into a tree.

The Huffman tree development is the main part of this project. For the Huffman code implementation, I decided to create a Minimum Heap using a simple node structure. The node takes in a few arguments: Character, frequency, left child and right child. This node structure allows the implementation of min heap to be created. Creating the min heap allows us to short the frequency table to a tree structure of the node values.

Tree structure allows the encoding to be processed very quickly. Following the Huffman coding definition allows either a min heap or max heap. A min heap allows the root to be the most common character of the frequency table.

Once the tree of the Huffman code is created, we can then use this tree to decode any encoded file and encode any clear text to a file as well. A method called encode_huffman and decode_huffman to decode/encode respectfully. The main idea for encoding is the most common characters in the nodes are used more commonly and have fewer binary numerical numbers to represent that frequency.