

Problem 3: Huffman Coding

This problem can be divided into two parts: the first **huffman_encoding()** and the second **huffman_decoding()**.

To solve this problem I used the following data structures:

A **Tree**, because I needed a left and right child to hold their corresponding nodes.

A **Priority Queue**, because the algorithm needs to prioritize from lowest to highest frequency.

A **Dictionary**, to hold on the frequency for each letter and the binary codes.

Space Time Complexity

For the time complexity of the **ENCODING** part I used a **loop** to build a dictionary of frequencies, then I used another **loop** to add nodes to a priority queue, then I used another **loop** for the priority queue till it's length reaches 1. Then did **traverse** the generated huffman tree thus adding $O(n)$ time complexity. Finally I used a **loop** to generate the encoded value from the given data.

All this process took: $O(n) + O(n) + O(n) + O(n) + O(n)$ which means 5 times $O(n)$ which can be described simply as **$O(n)$**

Time $\rightarrow O(n)$

Space $\rightarrow O(n)$, having n as the number of characters in the input data

For the **DECODING** I used a **loop** to iterate over the entire array of bits and check if the tree reaches a leaf with no left and right children, thus adding $O(n)$ time complexity.

Time $\rightarrow O(n)$

Space $\rightarrow O(n)$, because n is the number of bit characters in the input string