Huffman_Coding_Explanation

February 22, 2022

0.1 Problem Statement

This problem was concerned with building a function that encode and compress string data for purposes of transmission, and another function to decode the encoded message.

0.1.1 Design considerations

In order to do the encoding a Huffman-tree is required to be built. A `Node` class is defined that stores the various nodes and their binary codes in the huffman tree. A priority queue is also needed in order to build the tree. A dictionary for storing characters and their frequency is used to help build the priority queue. The priority queue has been implemented as a min-heap data structure to reduce time complexity during reinsertion operations that build the tree. Traversing the tree from root to all leaf nodes in order to generate the code was done recursively. The `Minheap` class contains methods for inserting and removing elements from the priority queue. Decoding the encoded message was a bit easier provided data and huffman tree were availible . This was achieved via iteration.

0.1.2 Time and Space complexity.

For decoding function we have one for loop depending on the characters in the message hence it's complexity is $O(n)$.

For encoding function we have $O(n)$ when building the frequency dictionary, another $O(n)$ when pushing into priority queue. Inside the priority queue if min-heap property is violated at the worst case which does not happen all the time there is $O(n)$. In order to generate the code we traverse the huffman tree in an inorder manner. This in turn is implemented via recursion hence the time complexity is $O(n)$. Finally we have one more for loop for building the encoded string. Hence the time complexity becomes $O(n) + O(n) + O(n) + O(n) + O(n) = O(n)$.

Space complexity in the decoding function is of linear nature dependent on encoded string length. i.e. $O(n)$. In the encode function there is a frequency dictionary of length n, a priority queue that ultimately has length of 1, there is a huffman tree of length $n2$, encoded_dict of length n, and encoded_data of length $n3$. Hence the space complexity is of a linear nature too. i.e. $O(n)$