Huffman Coding Assignment Report

CI583 – Data Structures and Operating Systems

Deferral 19th August 2022

Table of Contents

Encode complexity

Frequency table

In order to achieve an encoding method, we start by creating a frequency table. As we have to loop over each of the characters in the input in the input string using a for loop, the worst-case scenario complexity for this method would be the order of O(n) because if the length of the string ‘n’ grows, so the time that it takes to complete the loop grows in the same way that ‘n’ grows.

The Huffman tree

Using the ‘treeFromFreqTable’ method in ‘Huffman.java’ class, we start constructing the Huffman tree. Starting off by creating an empty priority queue followed by making a leaf node for every entry in the frequency table to be added to the new priority queue made in the start, that will allow us to retrieve the characters in the order of smallest to largest.

Next, we implement a while loop to go through the code, retrieving the left node and right node in order to combine them in a branch node, and put back in the queue. This sorting leads to adding and inserting values up to a point that we reach a greater value and other values will be shifted forward in the queue. Thus, resulting an O(n2) complexity as there would be a function of both insertion and removal from the queue.

“BulidCode” method

Constructing a map of characters and the code from the tree. We recursively call the ‘traverse’ method from the left and right nodes on the tree, by so receiving a map containing an ArrayList of all characters and their matching list of Booleans values that represent their code. The complexity remains O(n) as ‘n’ is the number of nodes as they all traverse get all codes from the tree.

Encoding the data

As we now have all the previous methods to generate the required code according to the given input string, the ‘encode’ method in the ‘huffman.java’ class can be completed.

We start by creating the frequency table so it can input the string in order to create the Huffman tree, while extracting the code map of characters and code from the tree, using ‘buildCode’ method to create the ‘treeFromFreqTable’ method. Then looping through for loop the input looking for each character in the map and adding that code to the list of character representing the data- by that encoding the data.

Despite that the ‘encode’ method only loops one character at a time, we have to take into account all other worst-case scenarios methods used to generate it.
The ‘treeFromFreqTable’ method has a complexity of O(n log n), based on the string input size and the use of while loop to iterate through the data.

Therefor, the ‘encode’ method also has an order of complexity O(n log n), and ‘n’ is the length of the input string.

1. Decode complexity

'TreeFromCode' method

To rebuild the Huffman tree, we require to start at the root of the tree and loop through each code to contact the branch and leaf nodes. That will be multiplied by the n of characters that needs to insert the tree.

Decoding the data

The 'decode' method is using a map of characters and all their codes and the encoded data as a list of Booleans. It is looping through the encoded data, and every time it reaches a leaf it has decoded, it adds on to the result and returns to the root of the tree, until it has gotten to the end of the data. And finally adding it to a string.

Although we loop over each Boolean encoded data is providing us with an O(n), the complexity is set by the worst performing method- 'treeFromCode' outcome of an order of complexity O(n2).

2. Data structures and algorithms in operating systems

Operating System acts as an interface between the user and hardware. Data structures allow an operating system to preform and engage in the most efficient way of managing the data resources stored or received for any given situation. However, input raw input data like values or sets of values is not necicerly secure and adjustable. This transforming process is called Hashing.

Hash functions

As there is a massive increase in the amount of data being processed by local and global data networks, we are always looking for ways to speed up data search that would take O(n) time when iterating through each file. One solution is a hash value, the output string generated by a hash function.

No matter what is the input, the generated output of strings in a hash function will always be the same length and they will be combined from a set of authorized charters defined by the hash function. For example, we can look at the 'hashcode()' method in Java that exchanges a string into a 32-bit integer using an algorithm. That will change all files using the hash function and store their locations, so we could simply search for any file using the same method to figure out it is location in the memory.

The key features of a hash function are that it is easy to compute the hash given a message, however the complecity to compute the message given the hash, and difficult to locate another message that would yield the same hash given a message (this is known as a collision).