Name: Edwin Kaburu

Date: 3/14/2022

CPSC-5600 Final Project Technical Paper

The problem that I am trying to solve is data compression, I want data to be compressed, or reduce in size and reconstructed to its original form without any data losses or clipping. Data compression is crucial in many aspects of computing in reducing the space occupied by data, but it has its fundamental difficulty in both theory and implementation. Currently there exist a diverse varieties of great compression algorithms, being utilized in real world applications, that range from being lossless and loosely. For the reasons, the goal is having compressed data reconstructed back to its original form without loss, I adapted the Huffman encoding in my project in solving the problem statement, as it is a lossless data compression algorithm.

My approach in the adoption of Huffman compression algorithm, is having characters of higher frequency with lower encoding, while characters of lower frequencies with a higher encoding instead of default encoding, eight bits, assigned to each character. This approach is optimal as it takes into consideration the occurrences of character within a sequence of characters. To accomplish this, I need a frequency table that contains the number of occurrences of unique characters within a sequence of characters. This is crucial as we want to reduce the sequence of many characters to only focus on the very few sequences of unique characters with a counter on their duplication through the sequence. The next process is utilizing the frequency table into forming a Huffman tree, whose interior nodes will contain the reduction or combination of the two smallest sums of occurrences, the greatest of the two will be on the right, the later will be on the left side.

The Huffman tree is important as it is the bridge or acts as gate keeper during compression and decompression phase of the data. Without the Huffman tree, data loss will be possible and finding an optimal encoding for the characters will be lost, achieving very minimal compression. Therefore, after forming a Huffman tree, the leaves will reference to an independent character from the frequency table, a traversal through the tree up to the character, a leaf, will generate a unique encoding code for that specific character, which will be used prominently during the compression and decompression phases.

With the optimal Huffman tree constructed, the decompression process is more streamlined, and the original raw data can be accessed, and every character will be injected with their new corresponding encoding code. The data at this point is compressed and will occupy few spaces when compared to its non-compressed form. My next approach in demonstrating lossless compression is to decompress the now compressed data. In this process, decoding will be possible due to the Huffman tree, mapping or viewing the encoding as path of traversal through the Huffman tree. The paths followed will lead to the leaves of the Huffman tree, that will contain the unique characters needed to reconstruct the compressed data back into its original form without any loss or clipping.

Briefly diving into the implementation of the project, I shall be discussing the how the frequency table is created from the raw data along with its time efficiency, how the Huffman tree and encoding is briefly implemented and their time efficiency. As discussed earlier the frequency tables, with contains unique characters along with their frequencies extracted from the raw

uncompressed data. Therefore, how it works, if that we loop through each character in the data and run a comparison checklist to investigate whether the character currently exists as an element within the frequency table. If the character does exist as an equivalent or duplicate, we can increase or increment the frequency of the unique character in the table, otherwise we add the entry to the table.

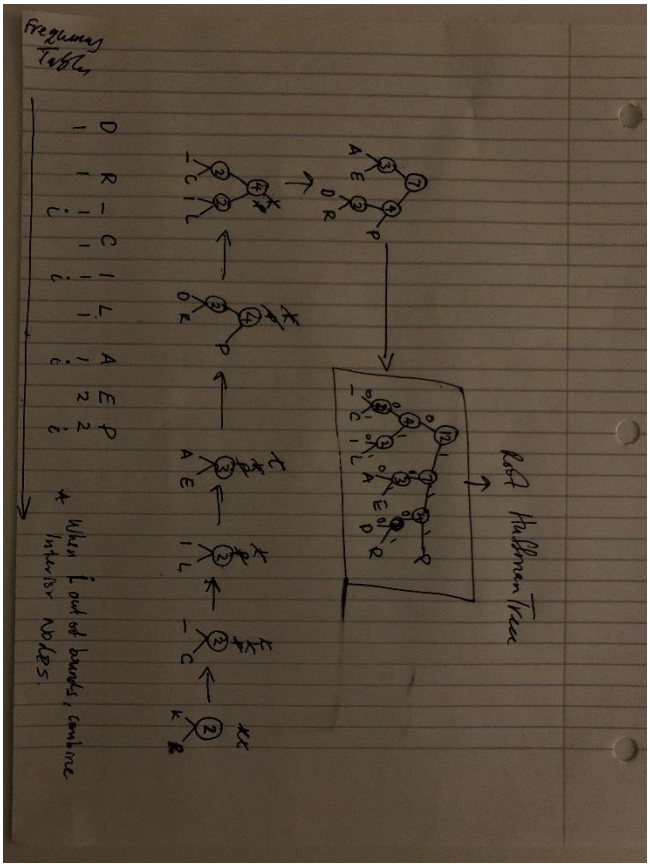
The counting frequencies is parallelized throughout the dataset, but it will not be unlimited parallelism because it will follow the Schwartz approach, therefore at some point where serial iteration will occur once a certain threshold have been reached. The efficiency relatively to latency will be, in the case of subset partitioning of the data, therefore

$$O\left(\frac{n}{p} + log_2\, p\right)$$

The work or cost to be carried out will be $O(n + p)$, the intuition behind this has to do with the fact there is a traverse through all entries in addition to the number of p threads created. Below is an example of frequency table generated, with the data "Apple Cider".

Symbol	Counts	Code: After Huffman Tree Generation
d	1	1100
r	1	1101
	1	000
C	1	001
i	1	010
l	1	011
a	1	100
e	2	101
p	2	111

The Huffman tree as discussed earlier, will combine the two smallest values or nodes to gather to form a node, where the rightmost will contain the largest of the two, while the leftmost will contain or hold the smallest of the two. This is crucial as the frequency table will be leaves, the path to it, will be determined by through mapping based on the conditions check. After a path and mapping have been established, the encoding can begin on each character based on its position and traversal, a right path has a weighted value of one, while the later to the left have a weight value of zero. This process will be serial, however since we will be iterating linearly through the, p, interior nodes, its latency and cost will be $O(p - 1)$. The code implementation will contain four pointers, i current frequency, k latest updated interior node, l for interior node available for comparison, and p placement position, due to their changing characteristics the example below will abstract these pointers away.



The encoding will finally compress the initial original data, it will go through each character in the data and update each of the characters with the new encoding generated to ultimately compress the data. Its latency and cost will be similar in equivalence to the counting frequencies. The decoding implementation will simplify loop through the compressed data entries which will be akin of a path that can be mapped with the Huffman tree to retrieve a character needed to reconstruct a compressed dataset. The implementation will be serial and therefore its cost and latency for the decoding process would be $O(n-1)+1$.

Apart from data compression, potential future applications where the Huffman encoding algorithms can be utilized could be in areas dealing with sequences of recurrence or occurrences in data such as multimedia file compressions, DNA and RNA encoding. The reasons I think, Huffman encoding could thrive in sequence of data with recurrences, have to do with frequencies. The frequency table as seen in implementation, is powerful in that a reduction of a set of common unique characters can be achieved, allowing for various transformations to applied on the data whether it is for DNA sequencing or encoding for data compression.