Huffman Algorithm Improvement

Apratim Gupta
Division of Computer Science, NSIT, Dwarka, New Delhi,
Delhi, India

DOI: 10.23956/ijarcsse/V7I6/0341

Abstract: The purpose of this paper is to propose a few theoretical improvements over the Huffman coding algorithm used for variable length encoding and data compression. The proposed algorithm works in a similar way as the Huffman algorithm, that is, it assigns a variable length code to each character. However, it makes use of the fact that texts in most languages are composed of words, which occur in varying frequencies and that not all combinations of letters in any language form valid words. To accommodate the improvements, the procedure is slightly modified. At the same time, it is still an improvement over Huffman encoding scheme, hence follows its rules. The binary representations allocated are unique, and no representation is the prefix of another.

Keywords: Huffman encoding, binary encoding, Huffman trie, Data Compression.

I. INTRODUCTION

Data Compression is one of the important processes used in the present world. They compress any form of data into a suitable form which can be stored in comparatively less space. This not only saves space, but it also saves time and resources which are used to transfer the data from one point to another. This research paper deals with a particular type of Data Compression technique, Huffman Algorithm. This paper comprises of a technique which can be used to compress a given text of data into binary codes, which can be regenerated using the decoding algorithm. The compression factor of the improved algorithm is theoretically greater than that of the Huffman’s. This algorithm can be used to compress data even further and can be used with the normal Huffman code with some modifications.

II. DESCRIPTION

Huffman encoding is a way to efficiently allocate variable length binary representations to a given set of symbols, depending on their frequency of occurrence. Thus characters with higher frequency are allotted smaller binary codes, and vice versa. It is often used in data compression, for the above listed property.

Texts in most languages are composed of words, which occur in varying frequencies. However the total numbers of words are much greater than the total number of characters. Also, there will always be combinations of characters which won’t be valid words. We utilise these two properties in improving Huffman encoding scheme.

1. Find the ‘N’ most frequent words. ‘N’ can be a function of the language used, the size or the kind of text to be compressed. Also note the frequencies of each of the words.
2. Form a Huffman trie using these words instead of characters, and their frequencies. Note, that the trie is formed assuming each word to represent a single character. Also, all the words in the text have not been covered. Let’s call it trie1. Also, store these words in a map.
3. A normal Huffman trie is formed using the entire text, with the leaves representing characters. Let’s call it trie2.
4. Now, a word is formed using the upper leaves of trie2 which does not exist in that language. For example in the English language, “aa” is a string which does not occur in the English language, and can be used.
5. This word is used as a marker to shift from trie2 to trie1, while decoding.
6. While decoding, if we come across the marker, then we jump to trie1, and traverse it using the following binary code till we hit a leaf. Otherwise, we simply use trie2 to decode.

Another way to improve upon Huffman works if the text size is large. In that scenario, the text is divided into parts, and a separate Huffman Trie is formed for each part. The reason this can perform better than a single Huffman trie for the entire text is because the binary code allocations for each part of the text would be optimised.

The most frequent words can further be divided across multiple Huffman tries, with a marker for each, in order to reduce the length of the binary representation for each word.

III. CHALLENGES

Since the above methods are theoretical, they present a number of challenges, such as:

1. While decoding, the Huffman trie is required, which also requires memory. In this case, we are relying on the use of multiple tries, which increases the space used, as all the tries would need to be stored. In addition, the leaves would store strings as opposed to just characters.
2. Time complexity can get affected as a map is used to determine if a word is in the list of most frequent words or not.
3. The task to find the most frequent words can be challenging as both the frequency as well as the length of the word should play a role in determining the size of the binary code allocated to it. Further if we use multiple tries, the lengths of the markers should also be pivotal in determining the trie in which a word ends up.
4. When following different tries for different parts of the text, it is difficult to determine the size of each part optimally. That is at what point will using a different trie negate the memory cost of storing it.

IV. PERFORMANCE

Since the improvements are largely theoretical and with the above challenges, they have not been used on large input texts. However using one additional trie for words, for relatively smaller sizes of inputs (8000-15000 words), the performance comes out to be between 5-8% better than simple Huffman encoding.

However this performance is not absolute because of the above challenges.

V. CONCLUSION

Data Compression is an extremely useful procedure and the above method is suggested as one of the ways to do it. This method, with careful considerations, can be further extended to data other than text as well.

REFERENCES