Abstract

Data compression plays a vital role in the storage systems. Different compression algorithms are applied to data depending on the situation. For example, if a set of data needs to be accessed frequently, the system will compress it with a simpler algorithm which allows the data to be read by the applications more efficiently. Conversely, if there is another set of space-consuming files that are rarely used, the system may suggest either deleting them or compressing them with a more advanced compression algorithm. In this research paper, we will delve into the theoretical foundations of data compressions, the compression algorithms as well as their applications.

Entropy-based methods

As proposed by Shannon's theory of entropy, data are characterized by their unpredictability and randomness. Thus, a standardized system of compression will not be expected to be an optimal solution. To cope with the unpredictable nature of data, Shannon's entropy was introduced. According to Wikipedia, "Shanno's entropy measures the information contained in a message as opposed to the portion of the message that is determined (or predictable)" (en.wikipedia.org). In other words, Shannon's entropy can be used to detect repetitive or redundant phrases in a text and prioritize the compression of these frequently occurring data, hence decreasing the size of the file more efficiently. This method is referred to as Huffman coding. Huffman coding assigns variable-length codes to input characters based on their frequencies. Frequently occurring characters get shorter codes, while rarer ones receive longer codes. Huffman coding is a prevalent method of compression due to its simplicity and efficiency. It is used in ZIP and gzip files and png for lossless compression.

Huffman coding example:

If we are tasked to compress the following data:

A: 45 occurrences

B: 13 occurrences

C: 12 occurrences

D: 16 occurrences

E: 9 occurrences

F: 5 occurrences

We need to first arrange them in an ascending order:

F(5), E(9), D(16), C(12), B(13), A(45)

Then, according to the fundamental rule of huffman coding, we need to combine the two terms with the lowest frequency, which are F and E in this case:

F(5) + E(9) = (F+E)(14)

The list becomes:

(F+E)(14), D(16), C(12), B(13), A(45)

Next we will need to combine C and B, which are the terms with the lowest frequency in the new list:

C(12) + B(13) = (C+B)(25)

The list becomes:

(F+E)(14), D(16), (C+B)(25), A(45)

From here we can see that the terms with the lowest frequency are (F+E) and D, so we will need to combine them:

(F+E)(14) + D(16) = (F+E+D)(30)

The list becomes:

(F+E+D)(30), (C+B)(25), A(45)

Repeating this pattern, we will get the following:

(F+E+D+C+B)(55), A(45)

Now, we will assign binary codes to each term (letters). Since A has the highest frequency, it will get the shortest code of 0. In the meantime, each term in (F+E+D+C+B) will get a 1.

Breaking the list down, we can see that it derives from (F+E+D)(30) and (C+B)(25). In this case, (F+E+D) has a higher frequency than (C+B), so each term in (F+E+D) will get a 1 while the terms in (C+B) will get a 0.

Further breaking down (C+B), we will get C(12), B(13). Applying the rule where the term with the higher frequency gets assigned a 1, while the one with the lower frequency gets a 0, the binary codes for C and B are 100 and 101, respectively.

At the end, we will get the following:

A: 0

B: 101

C: 100

D: 111

E: 1101

F: 1100

This will be the compressed form of each term in the original data. We can see that the letter A, with the highest frequency, has the shortest binary code representation while the letter F, with the lowest frequency, has one of the longest binary code representations.

Dictionary-based compression

This type of compression is usually more straightforward in terms of its functionality. Different from the entropy-based compression, dictionary-based compression aims to replace long, repetitive terms with shorter references, essentially building a dictionary of terms, or patterns, as the data is encoded. Dictionary-based compression is particularly efficient for data with repeated sequences, such as texts. An example of dictionary based compression will be the Lempel-Ziv-Welch (LZW). It replaces recurring terms of data with codes that refer to a dynamically built dictionary. The dictionary starts with single characters that exist in the string and adds longer sequences as it receives more inputs. Due to the flexibility of its dictionary, LZW compression is very versatile and can be applied to many situations not limited to textual compressions.

Lempel-Ziv-Welch Example

We are tasked to compress the following string using LZW:

ABABABA

The LZW dictionary will first include every single character that exists in the string. In this case, it will be A and B. So the first two numbers will be assigned to the two letters respectively:

A = 1

B = 2

Now, let's read the string by characters:

1. Read "A": Since it already exists in the dictionary, its numerical representation "1" will be added to the compressed sequence. (Current compressed sequence: 1)
2. Read "B": Same as the previous one, "2" will be added.

(Current compressed sequence: 1,2)

Additionally, we recognize the pattern AB here, so it will be assigned to the subsequent number code “3”.

Now, our dictionary will be:

A = 1

B = 2

AB = 3

3. Read “A”: “1” is added. (Current compressed sequence: 1,2,1)
4. Read “B”: We recognize that the pattern “AB” already exists in the dictionary, so code “3” will be added instead of “2”. (Current compressed sequence: 1,2,1,3)
5. Read “A”: “1” is added (Current compressed sequence: 1,2,1,3,1)
6. Read “B”: Same as step 4. “3” will be added. (Current compressed sequence: 1,2,1,3,1,3)
7. Read “A”: “1” is added. (Current compressed sequence: 1,2,1,3,1,3,1)

Lastly, the LZW compression of the string ABABABA yields 1,2,1,3,1,3,1 with the following dictionary:

A = 1

B = 2

AB = 3

Most may stumble on the 1,3 sequence, since according to the dictionary, (1,3) will yield A + AB = AAB. However, the LZW compression algorithm uses a different concept of addition rather than the literal interpretation of combining A with AB. The decoded form of (1,3), A + AB, conceptually means that when the sequence “A” followed by B is recognized as “AB” in the dictionary, you output the longer match (3) instead of individual components.

]The example above may cause confusion regarding the effectiveness of LZW compression. The fact is, the string provided above is too simple since it only contains basic 2-letter patterns. It can not show the full potential of LZW compression.

Conclusion

Two disparate compression algorithms are introduced in this research: Entropy-based and dictionary-based, with one example provided in each compression method. The huffman coding is assigned codes by the frequency of terms in a set of data while the Lempel-Ziv-Welch compression utilizes a dynamic dictionary to store the repetitive patterns in a given string or a set of data. Each compression method has its own application. From the examples provided, we can see that Lempel-Ziv-Welch compression is less performant when it comes to short pieces of data. However, its fundamental ideology of replacing adding and replacing recurring patterns with a dynamic dictionary will excel most compression methods when tasked to compress a long text.

Works cited

GeeksforGeeks. “LZW (Lempel–Ziv–Welch) Compression Technique - GeeksforGeeks.” GeeksforGeeks, 26 Apr. 2017, www.geeksforgeeks.org/lzw-lempel-ziv-welch-compression-technique.

“Huffman Coding | Dremio.” Www.dremio.com, www.dremio.com/wiki/huffman-coding.

Wikipedia Contributors. “Entropy.” Wikipedia, Wikimedia Foundation, 8 Nov. 2019, en.wikipedia.org/wiki/Entropy.