Compression

Tom Rochette <tom.rochette@coreteks.org>

January 12, 2020 — 183949b8

0.1 Context

Compression is the process of encoding information using fewer bits than the original representation[1]. It is the question of figuring out how to extract as much knowledge as possible and store it in as little space as possible. It is the ability to receive new data and to store it using existing knowledge.

0.2 Learned in this study

0.3 Things to explore

- How to compress a text the most?
- How can you discover information in a compressed source?
- Is it possible to devise a program that will generate the appropriate output based on a dictionary (key => value) in such a way that the program is smaller than the dictionary? Is this the same as writing compression of a file/dictionary?
- Everything can be considered as an index in a infinite dictionary of random digits and we are simply referring to specific points of that dictionary when we "create" functions such as trigonometric functions
- Compression can only occur if there is duplication/structure in data
- Strategies for declaring a dictionary
  - Declare the range of valid characters
  - Declare a list of valid characters

1 Overview

2 Notes

How can you efficiently store terabytes of data, with hundreds of gigabytes updated daily?

Assuming we start with an empty storage. Data could simply be added sequentially to the storage space as it comes in. During downtime, a period of low necessity to write to the storage space, a second process could go through the existing data in order to detect duplication and to compress it.

A simple strategy could be to split the existing storage space in two and see whether the two parts are the same. If they are, then you can reduce the storage space by half. If that is not the case (which is highly likely), then the first and second parts are split in half again. This is a basic divide-and-conquer approach. Assuming that we're storing at a byte level, we would then use the blocks 2, 3 and 4 and compare them against block 1, block 2 with 3 and 4, block 3 with 4. This process can be repeated until we reach the single byte level, at which point it is not very useful anymore.

This approach has many weaknesses (amongst others not listed here):

- It compares block of same sizes only, which means it cannot find a smaller block within a larger block and replace it with a common reference.

[1] https://en.wikipedia.org/wiki/Compression

- It only compares aligned blocks, that means that if block 1 could be found as part of blocks 2 and 3, it will not find that match.

Given that we now have numerous compression algorithms available, it would be more sensible to rely on one of them.

During the day we would want to use an online compression algorithm, most likely based on dictionary lookup with the most frequent fragments being stored in high bandwidth and low latency memory. It should be possible to make use of this online compression algorithm from the start, however it would require a certain amount of data before it has built any kind of meaningful lookup dictionary with frequency count.

Having online compression algorithm compress data as it comes in might prevent the offline compression algorithm from doing better compression. It may also mean that we need to decompress the data that was compressed using the online method so that the offline decompressor can compress it better. We can see online compression as short term memory, while offline compression would be long term memory.

One issue with this approach is retrieval.

3 See also

4 References

- https://en.wikipedia.org/wiki/Compression
- http://mattmahoney.net/dc/
- https://cs.fit.edu/~mmahoney/compression/rationale.html
- http://prize.hutter1.net/