Lossless Geometric Compression

Gleb Rusiaew

17 April 2019

1 Introduction

Arithmetic compression before rested in the problems of factorization. Geometric compression rested only in root-calculation possibility of computers. My paper apply a new method to an old problem. I have an implementation of this compression method in the Python 2.7 programming language.

This program does not claim to compress data in any way, however, it is possible in a significant number of cases. Compression of data chunk C takes place if the equation is true:

$$s(C) > s([\sqrt[n]{C}]) + s(C - [\sqrt[n]{C}]^n) + s(n) + 2 + 1$$

The function $s(x)$ means size of x in computer memory (in bytes).

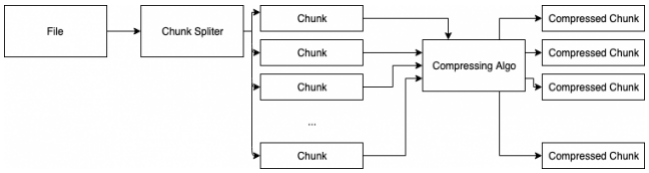
2 How it works?

Any number $a \in N_0$ can be represented as $[\sqrt[n]{a}]^n + (a - [\sqrt[n]{a}]^n)$, because:

$$[\sqrt[n]{a}]^n + (a - [\sqrt[n]{a}]^n) = [\sqrt[n]{a}]^n - [\sqrt[n]{a}]^n + a = 0 + a = a$$

Denote by $[x]$ the largest integer not exceeding x.

It mean that we can represent any a as vector $([\sqrt[n]{a}]; a - [\sqrt[n]{a}]^n; n)$.



3 Compession method

1. You must initialize the set t using this rule to compress the chunk c:

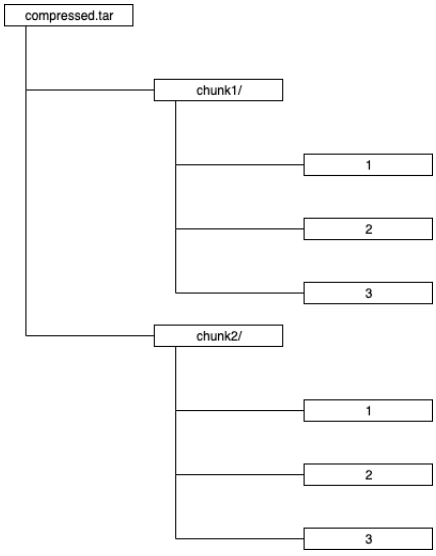
$$t_i = ([\sqrt[i]{c}], c - [\sqrt[i]{c}]^i, i)$$

2. Sort elements of t using $S(t_i)$ function (lower to higher).

3. Write t_i as compressed chunk (section 4).

4 How we write and store compressed chunks?

We can use the TAR to store a compressed chunk files (denote by c' a set of compressed chunks). Despite the ability to store files in my own format, I do not want to reinvent the wheel. This is the main reason why we use TAR. If necessary, you can modify the algorithm for recording data chunks.



This is a correspondence between the set elements and files:

chunk1.tar/2 $\leftrightarrow (c'_1)_2$
chunk2.tar/3 $\leftrightarrow (c'_2)_3$
...
chunk$N$.tar/i $\leftrightarrow (c'_N)_i$

5 Decompression

So, to decompress a array with compressed chunks (denote it by c') you must do this steps:

1. Initialize the set p using this rule.

$$p_i = (c'_i)_1^{(c'_i)_3} + (c'_i)_2$$

2. Write all elemants of set p in file p' from first to last.

Denote by f our original file. Now $p' = f$, because all chunks in p equals chunks in f. Since there were gradual transformations in the course of the paper, this does not need to be proved.

6 Realisation on Python 2.7 Conclusion

You can see the open source realisation of the LGC Algorithm and see updates news on the github repository `https://github.com/rusyaew/LGC`. If you want to add something new or bugfix the LGC, then send commits! And please recnese my paper (rusyaew@protonmail.com) if you can, that will help me a lot.