Lempel-Ziv with Integer Coders

Dominik Koeppl , TU Dortmund, Germany | dominik.koeppl@uni-dortmund.de

Host: SoBigData.it | Pisa, Italy

Text compression is the task to transform a text into a compressed form without losing information (i.e., the original text can be restored). One of the most common techniques applied by text compressors is the Lempel-Ziv 77 (LZ) factorization. We are interested in devising a compressor with the LZ factorization, since we believe that it is possible to improve the performance of common text compressors with a detailed analysis of the connection between the nature of the LZ factorization and sophisticated integer coders.

Factorization

The LZ factorization is a partitioning of a given text into factors. While scanning a text sequentially, we replace substrings with references to previous occurrences of the same substrings. A substring that is replaced with a reference is called a referencing factor. Each remaining character is called a literal factor. The idea is that referencing factors represent re-occurences that we can replace with a reference, i.e., an instruction for the decoder to restore the contents of the respective referencing factor.

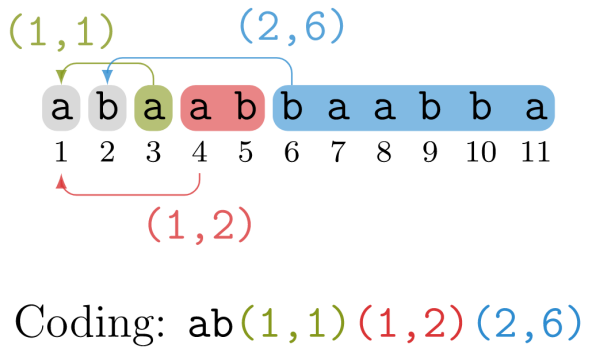

Figure 1: An LZ factorization of the text abaabbaabba. A factor is represented by a rounded box. Gray boxes represent literal factors, colored boxed represent referencing factors. A referencing factor has a backward reference consisting of a pair (j,ℓ), where j is the referred position and ℓ is the factor length. An arrow emerges from the box of a referencing factor pointing to its referred position. The final output consists of the literal factors and the references, which we here intersperse to build a single string called the coding.

Given a text T[1..n] of length n, an LZ factorization of T of size z consists of z factors $F_1 \cdots F_z$=T such that, for every $1 \le x \le z$, (referencing factor) F_x is equal to a substring starting before the factor F_x, or (literal factor) F_x is a character.

A referencing factor F refers to a starting position, called referred position, of a previous occurrence of F. Figure 1 visualizes the greedy LZ factorization choosing always the longest factor. Let us examine the greedy factorization on the example text T= abaabbaabba: Since the first two characters appear nowhere before, the factorization must put each of these characters in a literal factor. The third factor is a referencing factor of length one that refers to the first text position. The subsequent factor is also a referencing factor with the same referred position. Finally, let us have a look at the last factor: It refers to the substring T[2..7] although the factor covers T[6..11], i.e., the factor overlaps with its referred substring. This is actually a desired property, as it allows for higher compression ratios. For instance, we can factorize $T=a^n$ to a(1,n−1). Fortunately, the reference (2,6) corresponding to this factor is also decodeable: We first decode T[6..9] by copying the characters from T[2..5]. This restores the characters T[6..7] that we subsequently copy to restore T[10..11].

Our Goal

What we study in the following is the best way to store these factors in a compressed form. For that, we study how to encode factors. The standard coding scheme was introduced by Storer et. al [Storer82], where a referencing factor Fi with referred position j_i is replaced with a pair (j_i,ℓ_i) such that Fi=T[j_i..j_i+ℓ_i−1]. To store the LZ factorization of T, we need:

-one integer for the rank of the character of a literal factor, and

-two integers for the length and the referred position of a referencing factor.

The main problem is to minimize the number of bits needed to store these integers.

Compression Framework tudocomp

We can tackle this problem with the help of the framework tudocomp [Dinklage17]. It is a framework for devising and evaluating compressors and coders. The framework is written in modern C++14. It heavily embraces the C++ meta-programming features. Thanks to that, it allows for easy combinations of compressors and coders. The current version already contains a compressor, called lz77_lcp, that computes the greedy LZ factorization. This compressor can be parametrized by different coders. For that, the compressor calls the method encode of a coder with a value and the type of the value. By overloading this method with respect to the type, a coder can treat each type differently. The output of be lz77_lcp can be optimized by replacing each referred position with the distance to the starting position of its respective factor. These distances tend to be smaller than the referred positions.

Choice of Coders

For this article, we examine a 200 MiB prefix of the text collection pc-english of the Pizza&Chili corpus. A first hint for selecting a good combination of coders gives Figure 2 depicting the histograms of distances and lengths of the text collection pc-english. We see that the distribution of the lengths is monotonically decreasing after the length ten. If we do not allow the LZ factorization to produce referencing factors smaller than ten (which can be set with the parameter

threshold of the compressor lz77_lcp), we obtain a new distribution called cut lengths in the figure. The idea is to devise a coder that encodes legths with a universal code, whereas it encodes literals and distances with two separate statistical coders. The coder slecoder of tudocomp goes one step further and maintains different statistical coder techniques for the literal factors; it selects the coder of a literal factor based on its two preceding factors (if any).

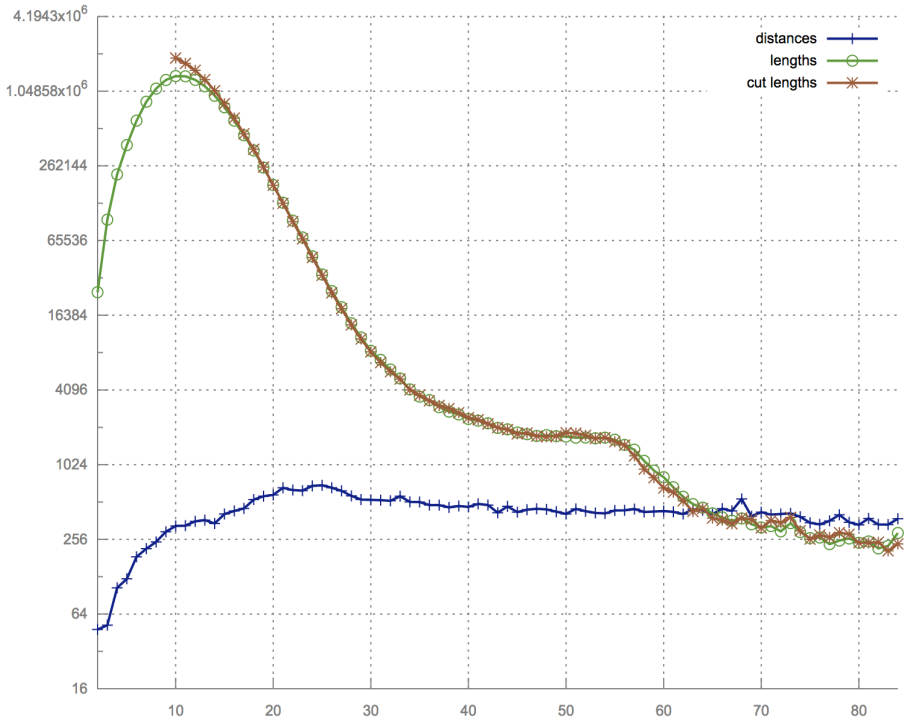

Figure 2: Histogram of the lengths and distances of the greedy factorization of the 200 MiB prefix of the collection pc-english. The x axis represents the values. We cut the values (which can range up to n=200 · 1024²) at 84. The y axis represents the number of occurrences of the respective value. It is in logarithmic scale (logarithm with base two). The function cut length are the lengths of the greedy factorization where we prohibited referencing factors smaller than ten.

Acknowledgements

The framework tudocomp is fostered by Patrick Dinklage and Marvin Löbel as part of their research assistant work. The contents of this article were worked out in a joint research with Paolo Ferragina and Antonio Frangioni during a stay at the university of Pisa from the 26th of March until the 12th of April. The author humbly thanks the organization team of SoBigData and their great support for making this kind of research possible.

References

[Dinklage17]
Patrick Dinklage, Johannes Fischer, Dominik Köppl, Marvin Löbel and Kunihiko Sadakane: Compression with the tudocomp Framework. SEA 2017: 13:1-13:22 doi: 10.4230/LIPIcs.SEA.2017.13

[Storer82]
J. A. Storer and T. G. Szymanski. Data compression via textual substitution. Journal of the ACM, 29(4):928–951, 1982. doi:10.1145/322344.322346.

[Ziv77]
J. Ziv and A. Lempel. A universal algorithm for sequential data compression. IEEE Transactions on Information Theory, 23(3):337–343, 1977. doi:10.1109/TIT.1977.1055714.