LZD と LZMW 分解の部分文字列圧縮について

クップル ドミニク

2023 年 10 月 18 日

概 要

LZD と LZMW は LZ78 から生じる文字列分解である．項の個数で考えた場合，LZ78 は $\Omega(\sqrt{n})$ の下界を持つが，LZD と LZMW の下界は $\Omega(\lg n)$ である．ただし n は文字列の長さである．整数アルファベットとして，LZ78 を決定的線形時間で計算できるが，LZD または LZMW を同様に計算できるかまだ明らかではない．本研究は，この問題を明らかにするとともに，LZD と LZMW の部分文字列圧縮問題についてのアルゴリズムを提案する．

キーワード： 可逆データ圧縮，Lempel–Ziv 78 分解，部分文字列圧縮，線形時間アルゴリズム

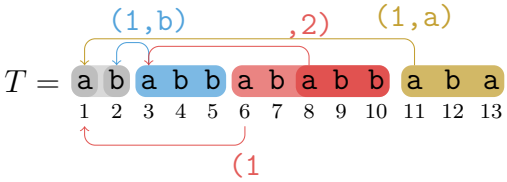
序論

可逆圧縮は，圧縮のうち，復元前の文字列を完全な形で取り出すことが可能な圧縮方法である．

文字列の可逆圧縮における一つの専門的分野，LZ78 分解 (Lempel–Ziv 78 分解) [11] がよく知られている．LZ78 は任意文字列 T に対し，T を項に分解する．このとき LZ78 分解は，各項がその項より前に出現する項に参照するように分解する．各項を参照している項と一つの追加された文字との二組に表現できる．ただし，前半を参照先と呼ぶ．すなわち，項を二組で表現した場合でも，元の文字列を復元できる．参照先を定めるため，様々な前の項のうちに，できるだけ長い候補を選択すべきである．

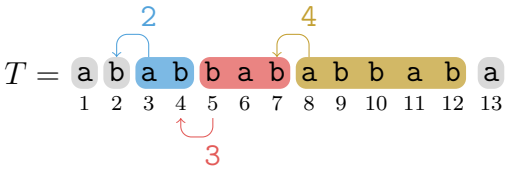
既に LZ78 の変種は複数定義されている．本研究では，その変種の中に，LZD [7] と LZMW [8] の 2 種類について取り上げる．共通の性質として，項は 2 つの参照先を持ち得る．そのために，LZD の項の二組の表現で，前半と後半は文字または参照先を格納できる．

例えば，$T = \mathtt{ababbababbabb}$ というテキストに対して，T の LZD 分解は $T = F_1 \cdots F_4$．ただし $F_1 = \mathtt{ab}$，$F_2 = \mathtt{abb} = F_1 \cdot \mathtt{b}$ ，$F_3 = \mathtt{ababb} = F_1 \cdot F_2$，と $F_4 = \mathtt{aba} = F_1 \cdot \mathtt{a}$ である．図 1 で項と参照を表現している．



LZD: (a,b)(1,b)(1,2)(1,a)

図 1: $T =$ ababbababbabb の LZD 分解．3 番目の項が 2 つの参照先を持つことを 2 つの矢印で表示された．



LZMW: ab234a

図 2: $T =$ ababbababbabb の LZMW 分解．参照先 $F_{x-1}F_x$ を持つ項はただ x で符号化されたお陰で，LZD の符号化より簡潔である．

選択肢が多いため，LZ78 より LZD では更に長い項を計算できるが，LZD の符号化は難しくなる．その一方で，LZMW は参照先について制限を持ち，LZ78 より符号化しやすい．より詳細には，LZMW で参照先は 2 つの連続する項になる．例えば，上記の例の文字列 T の LZMW 分解は $T = F_1 \cdots F_6$，ただし $F_1 =$ a，$F_2 =$ b，$F_3 =$ ab $= F_1F_2$，$F_4 =$ bab $= F_2F_3$，$F_5 =$ abbab $= F_3F_4$，と $F_6 =$ a である．図 2 で項と参照を示している．

LZ78 では，項の個数は $\Omega(\sqrt{n})$ の下界があるが，LZD と LZMW でその下界を突き破り，個数の下界は $\Omega(\lg n)$ になる ($T = \mathtt{a}^n$ の場合，LZD と LZMW の項の長さはそれぞれに 2 の冪とフィボナッチ数列のように大きくなる)．

LZD と LZMW どちらも最初に提案された構築アルゴリズムの時間は，$\Omega(n^{5/4})$ と証明されたが，他方で $\mathcal{O}(n + z \lg^2 n)$ 期待時間のアルゴリズムが提案された [1]．決定的アルゴリズムの場合，LZD を $\mathcal{O}(n \lg \sigma)$ 時間と $\mathcal{O}(n)$ 領域で計算できる [7]．

本研究では，決定的線形時間のアルゴリズムを提案する．目的は両方の分解を線形時間で計算するのみならず，任意の部分文字列 S に対して，S の分解を項の個数の線形的な時間で計算することを部分文字列圧縮と呼ぶ [4]．部分文字列圧縮問題とは，前処理でデータ構造を構築したあと，テクスト T の空間 $[i..j]$ をクエリー入力として，部分文字列 $T[i..j]$ の分解をいかに早く計算するかという問題である．素朴な方法は各部分文字列 S の分解を前処理で

格納し，定数時間でクエリーを答えられるが，領域は $\Omega(n^2)$ になってしまう．ここで，$\mathcal{O}(n)$ 時間と領域を持つ前処理を目指す．

本論では，$\mathcal{O}(n)$ 時間でデータ構造を構築したあと，そのデータ構造で任意の部分文字列 S に対して，S の LZD または S の LZMW を項の個数の線形的な時間で計算する．

予備知識

Σ をアルファベットとし， Σ^* の要素を文字列と呼ぶ．文字列 T の長さを $|T|$ と表記する．文字列 $T = XYZ$ について X, Y, Z をそれぞれ文字列 T の接頭辞，部分文字列，接尾辞と呼ぶ．$T[i]$ は T 中の i 文字目の文字，$T[i..j]$ は T の i 文字目から j 文字目までの部分文字列を表す．$j = |T|$ のとき，$T[i..j]$ は開始位置 i を持つ接尾辞であり，$T[i..]$ とも書く．ϵ は空文字列を示す．

以降，文字列 T を入力として固定する．$\sigma := |\Sigma|$ と $n := |T|$ は アルファベットサイズと $T[1..n] = T$ の長さをそれぞれ示す．

文字列分解　$T[1..n] = F_1 \cdots F_z$ を $F_1, \ldots, F_z$ の項に分解する LZD は，以下の状況を満たす．各 $x \in [1..z]$ に対して $F_x = G_1 \cdot G_2$，ただし $G_1, G_2 \in \{F_1, \ldots, F_{x-1}\} \cup \Sigma$ かつ G_1 と G_2 は，それぞれに $T[\mathsf{dst}_x..]$ と $T[\mathsf{dst}_x + |G_1|..]$ の最長接頭辞である．

LZMW 分解は $T[1..n] = F_1 \cdots F_z$ を $F_1, \ldots, F_z$ の項に分解する．ただし，各 $x \in [1..z]$ に対して F_x は $\{F_{y-1}F_y : y \in [2..\mathsf{dst}_x - 1]\} \cup \Sigma$ の中にある $T[\mathsf{dst}_x..]$ の最長接頭辞である．

前処理

LZ78 項 u の二組は参照先 v と文字 c に成り立ち，$u \to_c v$ のような写像で表現できる．その写像は trie で表現し， LZ trie を呼ばれる．LZ trie は接尾辞 trie の部分木であると知られ，$\mathcal{O}(n)$ 領域と構築時間を持つ接尾辞木 ST [5, 10] の中に LZ trie を動的に表現できる [9, Sect. 3]．各項は LZ trie の頂点で表現されているが，LZ trie の頂点は必ずしも ST の頂点で表現されていない．この場合，その頂点は ST の辺の中に表現されて，陰の頂点と呼ぶ．

その一方で，LZD と LZMW の項を trie で表現できない理由は，項が 2 つの項の縦続で成り立つ可能性がある．従って，[9] の LZ78 構築アルゴリズムを直接に応用できず，下記の 2 つの工夫されたデータ構造を利用する．

補題 1 ([3])．任意の頂点をマークし，最低のマークされた頂点を検索し，両方を定数時間でできるデータ構造が存在する．このデータ構造が ST の頂点の上で $\mathcal{O}(n)$ 時間で構築できる．

ST 頂点 v の文字列の深さ (string depth) は，ST から v までの経路を辿りながら，辺のラベルの長さの和となる．任意の葉 λ と文字列の深さ d に対して，文字列の深さ d を持つ λ の先祖の出力に対する文字列の重み付き祖先クエリ (weighted ancestor query) と呼ぶ．

補題 2 ([2, 6])**.** ST に対して，文字列の重み付き祖先クエリを定数時間で答えるデータ構造が存在する．そのデータ構造の構築時間は線形時間である．

アルゴリズム

LZD　　ST の上で，補題 1 と 補題 2 のデータ構造を前処理で構築する．各頂点 v に整数を格納させる．その整数は v に入る辺の中に最低の陰の頂点を示し，v の発見数と呼ぶ．

クエリー空間 $[i..j]$ とした場合，以下のように LZD を計算する．各項 F_x に対して，接尾辞番号 dst_x を持つ葉の最低マークされた先祖 v を計算する．ただし，dst_x は F_x の開始位置を示す．v の文字列の深さと v の発見数の和は ℓ_v とすると，接尾辞番号 $\mathsf{dst}_x + \ell_v - 1$ を持つ葉の最低マークされた先祖 w を計算する．w の文字列の深さと w の発見数の和は ℓ_w とすると，F_x の長さが $\ell_v + \ell_w$ に等しい．ST で F_x に対している LZ 頂点を挿入するため，F_x を接頭辞として持つ頂点のうちに，最高の頂点 u を検索する．u の文字列の深さは $|F_x|$ であれば，u は F_x の LZ 頂点を表現する．その一方で，F_x の LZ 頂点は陰の場合，u の発見数で表現するため，発見数を調整する．いずれの場合も，u をマークする．(後者の場合，すでに u がマークされた可能性があるが，それは問題ない.）最後に，F_{x+1} の計算を続ける．

計算量　　各項に対して，２つのマークされた最低の先祖を検索し，u を見つけるため一つの文字列の重み付き祖先クエリを行い，u をマークし，u の発見数を調整する．各ステップは定数時間で行うことができるので，時間計算量は項の個数の線形である．

LZMW　　前処理は前述のように行う．アルゴリズムについてクエリーとの違いは接尾辞番号 dst_x を持つ葉の最低マークされた先祖 v を検索しても十分である．そして，F_x の長さは v の文字列の深さと v の発見数の和になる．最後に，F_x の変わりに $F_{x-1}F_x$ を ST で検索する．以上のような頂点 u を見つけたあと，u をマークし，u の発見数を調整する．

定理 1. 任意の区間 $[i..j] \subset [1..n]$ に対して，LZD または LZMW を $\mathcal{O}(z)$ 時間で計算できる．ただし， z は $T[i..j]$ のぞれぞれの項の個数を示す．前処理は $\mathcal{O}(n)$ 時間で行うことができる．

謝辞

本研究は JSPS 科研費 JP23H04378 と JP21K17701 の助成を受けたものです.

参考文献

[1] G. Badkobeh, T. Gagie, S. Inenaga, T. Kociumaka, D. Kosolobov, and S. J. Puglisi. On two LZ78-style grammars: Compression bounds and compressed-space computation. In *Proc. SPIRE*, volume 10508 of *LNCS*, pages 51–67, 2017.

[2] D. Belazzougui, D. Kosolobov, S. J. Puglisi, and R. Raman. Weighted ancestors in suffix trees revisited. In *Proc. CPM*, volume 191 of *LIPIcs*, pages 8:1–8:15, 2021.

[3] R. Cole and R. Hariharan. Dynamic LCA queries on trees. *SIAM J. Comput.*, 34(4):894–923, 2005.

[4] G. Cormode and S. Muthukrishnan. Substring compression problems. In *Proc. SODA*, pages 321–330, 2005.

[5] M. Farach-Colton, P. Ferragina, and S. Muthukrishnan. On the sorting-complexity of suffix tree construction. *J. ACM*, 47(6):987–1011, 2000.

[6] P. Gawrychowski, M. Lewenstein, and P. K. Nicholson. Weighted ancestors in suffix trees. In *Proc. ESA*, volume 8737 of *LNCS*, pages 455–466, 2014.

[7] K. Goto, H. Bannai, S. Inenaga, and M. Takeda. LZD factorization: Simple and practical online grammar compression with variable-to-fixed encoding. In *Proc. CPM*, volume 9133 of *LNCS*, pages 219–230, 2015.

[8] V. S. Miller and M. N. Wegman. Variations on a theme by Ziv and Lempel. In *Combinatorial Algorithms on Words*, pages 131–140, Berlin, Heidelberg, 1985.

[9] Y. Nakashima, T. I, S. Inenaga, H. Bannai, and M. Takeda. Constructing LZ78 tries and position heaps in linear time for large alphabets. *Inf. Process. Lett.*, 115(9):655–659, 2015.

[10] P. Weiner. Linear pattern matching algorithms. In *Proc. SWAT*, pages 1–11, 1973.

[11] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Trans. Information Theory*, 24(5):530–536, 1978.