トライと三分木を使った高速化技法
LZ78符号によるファイルの圧縮と改良
前編

広井 誠

データ圧縮アルゴリズムにはいろいろな方法がありますが，多くの圧縮ツールでよく使われている方法にLZ符号があります．LZ符号はJ.ZipとA.Lempelが開発した「辞書に基づく符号化方式（辞書法）」の一つで，入力された記号列を辞書に登録し，その辞書を使って符号化を行う方法です．

LZ符号には多数のバリエーションが存在しますが，これらはLZ77符号とLZ78符号の二つに大別されます．LZ77符号は1977年に，LZ78符号は1978年に開発されました．LZ77符号はスライド辞書法，LZ78符号は動的辞書法と呼ばれています．よく使われている圧縮ツールであるLHAやzipはLZ77符号とハフマン符号を組み合わせた方法です．

一般に，圧縮率が高いアルゴリズムはデータの圧縮に時間がかかります．実際，LZ77符号はLZ78符号よりも圧縮率が高くなる場合が多いのですが，辞書の探索に時間がかかるため，符号化の処理速度はLZ78符号よりも遅くなります．圧縮ツールにはいろいろな用途があり，圧縮率の高さよりも符号化や復号の処理時間が重要になる場合もあります．

とくに，符号化の処理時間が問題になる場合，LZ77符号よりもLZ78符号のほうが適しています．LZ78符号はトライという木構造を用いることで，辞書を高速に探索できるため，符号化の処理速度はLZ77符号よりも高速です．トライにはいくつかの実装方法がありますが，Ternary Search Treeという三分木を用いると，辞書の探索をさらに高速化することが可能です．

LZ78符号の一つであるLZW符号は，米国Unisys社が特許をもっていましたが，2003年6月に米国で，日本では2004年6月に失効しました．LZW符号は優れたデータ圧縮アルゴリズムです．これを機会にLZ78（LZW）符号を見直してみましょう．

そこで，本稿ではLZ78符号の基本を解説し，高速な実装方法について詳しく説明します．

LZ78符号（LZW符号）

● 辞書の作成方法で効率が変わる

LZ77符号と同じようにLZ78符号にも多数のバリエーションが存在します．その中でもっとも基本的な符号が1984年にT.Welch氏によって開発された「LZW符号」です．本稿ではLZW符号を使って説明します．

LZ77符号はスライド窓を辞書として使用します．スライド窓の大きさは一定なので，記号を読み込むたびに古い記号を捨てていかなくてはなりません．つまり，LZ77符号は局所的な辞書を構成していると考えることができます．したがって，同じ記号列がファイル内に分散している状態では，それを効果的に圧縮することはできません．

この欠点は，スライド窓を大きく設定することで改良できます．ところが，辞書の位置情報や一致長を表す符号長が長くなるので，単純にスライド窓を大きくするだけでは，圧縮率を高くすることはできません．また，辞書の探索処理も時間がかかるため，符号化になおさら時間がかかることになります．このため，スライド窓を大きくすることは現実的ではないとされています．

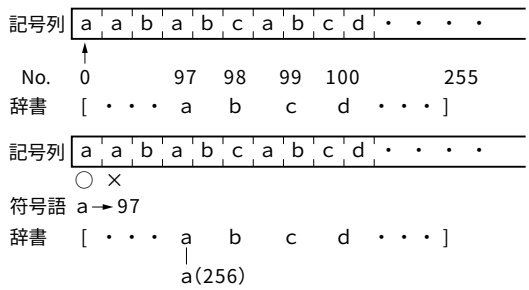
なお，LZ77符号の改良方法については，参考文献(4)で詳しく説明しています．興味のある方は参考にしてください．

これらの問題点を回避するため，LZ78符号ではスライド窓の使用をやめて，これまでに出現した記号列を辞書に登録することで大域的な辞書を作成します．この辞書の作り方によっては，メモリ・サイズが大きくなったり探索に時間がかかってしまいますが，LZ78符号では「トライ（trie）」という木構造を用いることで，効率的な探索処理を実現しています．トライについては，あとで詳しく説明します．

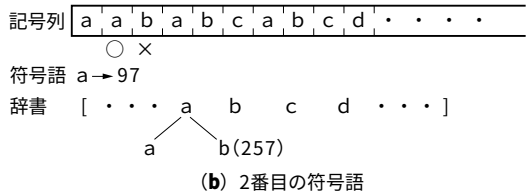
● LZW符号の符号化

それでは，記号列“aababcabcd”をLZW符号で符号化するようすを具体的に説明します．

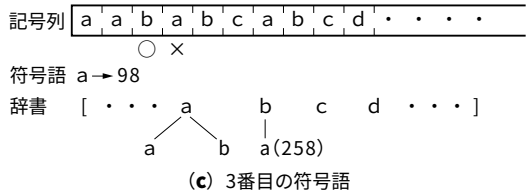
LZW符号では，最初に256種類の記号すべてを辞書に登録しておきます．たとえば，記号aは97番目でbは98番目になります．この状態から記号列を読み込んで符号化を行います．**図1**を見てください．



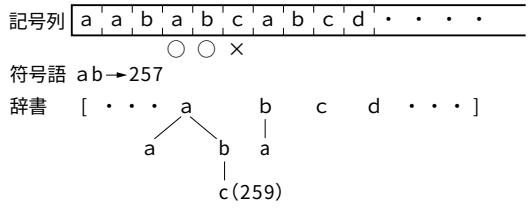
(**a**) 最初の符号語



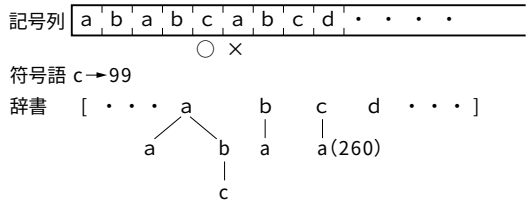
(**b**) 2番目の符号語



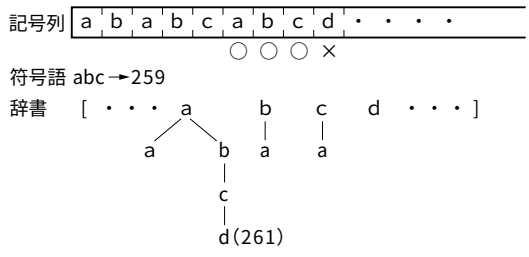
(**c**) 3番目の符号語



(**d**) 4番目の符号語



(**e**) 5番目の符号語



(**f**) 6番目の符号語

図1 LZW 符号の符号化1

まず，辞書の中から最長一致列を探すため，記号を読み込んでトライを探索します．最初の記号 a は辞書の 97 番目に登録されています．次の記号 a を読み込みますが，“ aa ”は辞書に登録されていないので，最長一致列は“ a ”となります．

そこで，**図1**(**a**)のように辞書番号 97 を符号語として出力します．それから，不一致になった記号 a を辞書 97 の後ろに追加します〔**図1**(**a**)〕．このときの辞書番号は 256 となります．

次に，不一致となった記号 a から最長一致列を探索します．**図1**(**b**)を見てください．次の記号 b を読み込んで辞書を探索しますが，“ ab ”は辞書に登録されていないので，“ a ”が最長一致列となります．辞書番号 97 を符号語として出力し，記号 b を辞書 97 の後ろに追加します．このときの辞書番号は 257 となります．

同様に，記号 b からの最長一致列を探索すると“ b ”が得られます．符号語を出力して辞書に登録します〔**図1**(**c**)〕．その次の最長一致列は，97→257 とたどり“ ab ”となります．そこで，辞書番号 257 を出力し，不一致記号 c を 257 の後ろに追加します〔**図1**(**d**)〕．

次は記号 c から始まりますが，最長一致列は“ c ”となるので，符号語 99 を出力して辞書 260 を追加します〔**図1**(**e**)〕．次の最長一致列は，97→257→259 とたどって“ abc ”となります．辞書番号 259 を符号語として出力して，辞書 261 を追加します〔**図1**(**f**)〕．

このように LZW 符号は，記号を読み込むに従って，辞書に登録される記号列が増えるので，効率的な圧縮が可能になります．

● LZW 符号の復号

次に，復号の手順について説明します．**図2**を見てください．

最初は符号化と同様に 256 種類の記号を辞書へ登録します．まず，符号語 97 を読み込み a と復号します．復号した辞書番号を記憶しておいて，次の符号語を復号します．符号語は 97 なので“ a ”と復号できます．このとき，復号した記号列の先頭記号を，直前に復号した辞書番号の後ろへ追加します．この場合は，辞書 97 の後ろへ記号 a を追加します．このときの辞書番号は 256 となります〔**図2**(**a**)〕．

次の符号語は 98 なので“ b ”と復号して，辞書 97 の後ろに記号 b を追加します．このときの辞書番号は 257 となります〔**図2**(**b**)〕．次の符号語は 257 なので，辞書を逆順に 257→97 とたどり，記号列“ ab ”と復号します．そして，辞書 98 の後ろに記号列の先頭記号 a を追加します〔**図2**(**c**)〕．

次の符号語は 99 なので，記号“ c ”と復号できます．そして，辞書 257 の後ろに記号 c を追加します．このときの辞書番号は 259 となります〔**図2**(**d**)〕．次の符号語は 259 なので，節を逆順に 259→257→97 とたどり，記号列“ abc ”と復号します．そして，辞書 99 の後ろに記号 a を追加します〔**図2**(**e**)〕．

このように，LZW 符号では符号語を復号していく過程で，辞書が復元されていきます．この例ではうまくいきましたが，注意しなければならない点が一つあります．記号列“ abababab ”