第14回情報理論とその応用シンポジウム
The 14th Symposium on Information Theory
and Its Applications (SITA '91)
Ibusuki, Japan, December 11-14, 1991

高頻度隣接の連結に基づいたデータ圧縮法

The data compression based on concatenation of frequentative code neighbor

中村　博文*1　村島　定行*2

Hirofumi NAKAMURA　Sadayuki MURASHIMA

Abstract : A parsing algorithm for sequential data compression is given. And a coding algorithm and a decoding algorithm based on this parsing is given. The analysis for parsing is to repeat the opration to regard frequentative neighbor of two code as one new code.

1. まえがき

一次元データの圧縮において、記号列の分解情報を利用する圧縮法がＺｉｖとＬｅｍｐｅｌによって提案(1)(2)され発展(3)(4)(5)(6)されている。いずれも入力データの確率構造を学習しながら圧縮していく適応的な方法である。また、ひとつの切り出しごとに圧縮符号が決まるが、圧縮符号が自己参照的であるため、復号側へ符号表を伝えなくても、復号側では復号しながら符号表を再現できるという特徴を持っている。更に、漸近的最良性が証明されている方法(1)(2)(5)(6)もある。

実用されているＬＺＷ法(3)とその基になった増分分解法(2)について概要を述べると次の通りである。

増分分解法では、ひとつの切り出しを、過去に登録した（過去に切り出しした）もっとも長い一致記号列と１記号とを連結した内容で行い、圧縮符号は一致記号列に登録時につけた番号と１記号の順序数との２つで構成している。

ＬＺＷ法では、ひとつの切り出しを、過去に登録した（切り出し記号列とつぎの記号とを連結した内容が登録されている）もっとも長い一致記号列で行い、圧縮符号は登録内容につけてある順序数ひとつで構成している。

しかし、以上の方法は次の点が目につく。

（１）類似の内容の符号表への登録

被圧縮記号列の中に繰り返し現れる記号列がその中間で様々に切り出されて、先頭や末尾が少しずつ異なる記号列が多数登録されている。

（２）現れない記号列の符号表への登録

登録の時点で使われる可能性が不明なことから、その後現れない記号列も登録している。

これらを次のように解決することを試みた。

（１）類似の内容の登録の回避

被圧縮記号列中に繰り返し現れる記号列をできるだけひとつの内容として登録する。

そのために先読みをする。ここでは適応的ではなくなるが、ファイルの終わりまで読んで登録と切り出しのための記号列の解析を試みる。２つの記号が隣接して出現する回数が多い隣接から順にひとつの記号と見なし、その内容を基に自己参照的になるように切り出しを行う。

（２）現れ得る記号列の登録

上記の（１）の先読みした上での出現回数が多い隣接の登録を、出現回数が２以上について行えば、後側で絶対に現れない内容の記号列を登録することはない。

*1 都城工業高等専門学校、都城市
Miyakonojo National College of Technology,
Miyakonojo-shi,885 Japan
*2 鹿児島大学工学部電子工学科，鹿児島市
Faculty of Engineering, Kagoshima Univ.,
Kagoshima-shi,980 Japan

2．解析と符号化

以下４元符号の圧縮例を用いて説明する。

2．1　使用する用語等

圧縮したい記号列

M_0＝｜a b a b c b a b a b a a a a a a a a｜

があるとする。記号列は、使用記号の集合

A_0＝｛a,b,c,d｝

の要素の、０回以上の出現による並びである。

A_0の全要素に a＜b＜c＜d なる大小関係を仮定する。また、更に記号が必要なときには、新たな記号としてA_0の最大の要素に続けて d＜e＜f＜g のような記号をいくらでも使用できるものとする。

２記号が隣合うこと又はこの２記号の組を隣接という。隣接 abを単に a b と記述する。

a b と　b a とは異なる隣接である。

記号列中で２記号の隣接が現れる回数を隣接数という。但し、M_0における a a のように同じ記号の隣接を数える場合は、記号列中でひとつの記号を２回は数え上げないことにする。

M_0で a a の隣接数は３である。

隣接数が最大の隣接を最頻の隣接という。最頻の隣接をなす隣接は複数のこともありうる。

隣接する２つの記号 a b を連結して、ひとつの記号とみなすとき、 $\overset{\frown}{a\ b}$ のように記述する。これを２進木とも見なす。

2．2　解析

記号列の解析は次のようにして行う。

（１）記号列中の最頻の隣接からひとつ選び、記号列中のその隣接をすべて連結する。連結は新たなひとつの記号とみなす。

（２）（１）を、最大の隣接数が２以上である間繰り返す。

以下例を用いて説明する。

M_0では隣接数４の隣接 a b と b a がもっとも頻繁に現れている。どちらでもよいがここでは a b を選ぶことにする。

M_0の a b を連結してひとつの記号にすると次のようになる。

M_1＝｜$\overset{\frown}{a\ b}$ $\overset{\frown}{a\ b}$ c b $\overset{\frown}{a\ b}$ $\overset{\frown}{a\ b}$ a a a a a a｜

a b を連結してひとつの記号とみなしたことを符号表に次のように憶えておく。

T_1＝｜(a b $\overset{\frown}{a\ b}$ nil)｜

nil は a bの連結にまだ新しい記号を割り当てていないことを表す。

次に、隣接数３の隣接　a a を連結して次のようになる。

M_2＝｜$\overset{\frown}{a\ b}$ $\overset{\frown}{a\ b}$ c b $\overset{\frown}{a\ b}$ $\overset{\frown}{a\ b}$ $\overset{\frown}{a\ a}$ $\overset{\frown}{a\ a}$ $\overset{\frown}{a\ a}$ a｜

T_2＝｜(a b $\overset{\frown}{a\ b}$ nil)(a a $\overset{\frown}{a\ a}$ nil)｜

更に、隣接数２の $\overset{\frown}{a\ b}$ $\overset{\frown}{a\ b}$ を連結して次のようになる。

M_3＝｜$\overset{\frown}{\overset{\frown}{a\ b}\ \overset{\frown}{a\ b}}$ c $\overset{\frown}{\overset{\frown}{a\ b}\ \overset{\frown}{a\ b}}$ $\overset{\frown}{a\ a}$ $\overset{\frown}{a\ a}$ $\overset{\frown}{a\ a}$ a｜

T_3＝｜(a b $\overset{\frown}{a\ b}$ nil)(a a $\overset{\frown}{a\ a}$ nil)

(a b $\overset{\frown}{a\ b}$ $\overset{\frown}{a\ b}$ nil)｜

隣接数２以上の隣接はなくなったので、まとめる作業を終える。

2．3　切り出しと符号化

M_3、T_3の情報を構造を持たない記号の列として符号化する方法は一通りではないがその中でも自己参照的な符号化をひとつを挙げる。

例えば次のように符号化できる。

F_1＝｜1 2 5 3 2 6 1 1 7 7 1｜

F_2＝｜1 1 0 0 0 0 0 0 1 0 0 0 0 0｜

これらをまとめて次のようにも符号化できる。

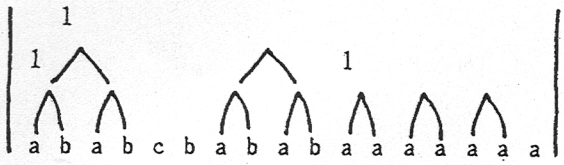
F＝｜1101020503020610101070701｜

F$_1$、F$_2$、Fは次のような手順で決定した。

T_3中のすべてのまとめ方

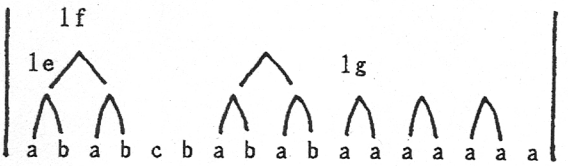
$\overset{\frown}{a\ b}$、$\overset{\frown}{a\ a}$、$\overset{\frown}{\overset{\frown}{a\ b}\ \overset{\frown}{a\ b}}$

が、M_3に現れる最初の所に印１を付ける。

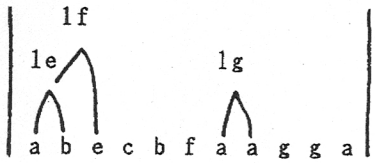


この例のように、あるまとめ方が別のまとめ方の部分になっていてもよい。

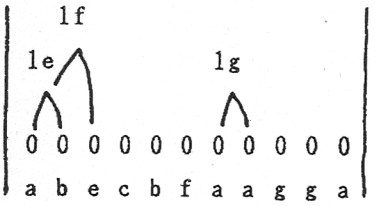
ここで各記号を左から、左下、右下、そこの順で再帰的に探索し、出会う印１の所に新たな記号を順次付けていく。これはT_3中のまとめ方のそれぞれに記号を割り当てていることに等しい。



印１の付いていないくくり方について、T_3を見ながら新しく割り当てた記号で置き換える。



印１の付いていないすべての a から g の記号に印０（ゼロ）を付ける。葉にのみ０が付く。



これを木や葉の列とみなし葉のみ並べた

| a b e c b f a a g g a |

の中の記号をその順序数で置き換えてF_1を得る。

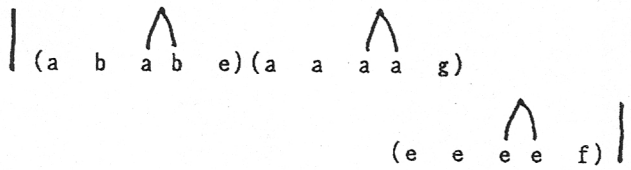
印１、０からなる木や葉の列の部分を、１が木を作る演算子で０が葉となる被演算子である構文木とみなして、前置記法で表してF_2を得る。

F_2の０の後に対応する葉を並べた

| 110a0b0e0c0b0f10a0a0g0g0a |

の中の記号をその順序数で置き換えてＦを得る。

T_3をF_1のように書き換えると次のようになる。



符号化は例えばF_2の要素を各１ｂｉｔで、F_1の要素を各⌈ｌｏｇ Ｃ⌉ ｂｉｔでできる（但しＣはその時点で使われ得る記号の数）。

３．　符号化と復号化のアルゴリズム

符号化アルゴリズムをPascal風に記述し図１．に示す。各記号は次の意味で用いている。

Ｋ：被圧縮記号列の記号数（Ｋ元符号）

Ｃmax：その時点での最大の符号の順序数

readfile(Ｍ)：Ｍに被圧縮ファイルを読み込む

maxn(Ｍ,ci,cj)：Ｍの最頻の隣接から適当な１組　ci cj を与え、又その隣接数を与える

enter(Ｔ,cl,cr,ck,nil)：clが表す記号列とcrが表す記号列の連結記号列（ckが表す）にnilを割り当てて符号表Ｔに登録する

change(Ｍ,ci,cj,ck)：Ｍ中での隣接　ci cj をすべて ck に変える

length(Ｍ,i)　：Ｍの要素数を与える

element(Ｍ,i)：Ｍのｉ番目の要素を与える

writetreecode(c)：木構造の指示符号ｃ（０か１）を出力する

writecode(c)　：符号ｃを出力する

assignedcode(Ｔ,c)：符号表Ｔ中でｃに割り当てられた符号を与える

left(c)　：木ｃの左側の枝を与える

right(c)：木ｃの右側の枝を与える

reenter(Ｔ,cl,cr,ck,c)：以前のcl、cr、ck、nilでの登録は削除して、clが表す記号列とcrが表す記号列の連結記号列（ckが表す）に符号ｃを割り当て符号表Ｔに再登録する

```
procedure writetree(c);
  begin
    if c<K then begin
      writetreecode(0);                    { make F2 or F }
      writecode(c);                        { make F1 or F }
    end else if assignedcode(T,c) ≠ nil then begin
      writetreecode(0);                    { make F2 or F }
      writecode(assignedcode(T,c));        { make F1 or F }
    end else begin
      writetreecode(1);                    { make F2 or F }
      writetree(left(c)); writetree(right(c)); Cmax:=Cmax + 1;
      reenter(left(c),right(c),c,Cmax);
    end;
  end;
procedure code;
  begin
    readfile(M);
    while maxn(M,ci,cj) >= 2 do begin
      enter(T,ci,cj,ci^cj,nil); change(M,ci,cj,ci^cj);
    end;
    Cmax:=K - 1;
    for i:=1 to length(M) do writetree(element(M,i));
  end;
```

図１．符号化アルゴリズム

次に復号化アルゴリズムを図２．に示す。各記号は次の意味で用いている。

Ｋ、Ｃmax：上記と同様

writetreecode(c)：木構造の指示符号ｃ（０か１）を入力する

writecode(c)　：符号ｃを入力する

write(c)　：順序数がcとなる記号を出力する

left(T,c)：符号cが表す記号列を構成した左側の記号列の符号を与える

right(T,c)：符号cが表す記号列を構成した右側の記号列の符号を与える

enter(T,cl,cr,ck,c)：符号clとcrの連結ckに符号cを割り当てて符号表Tに登録する

```
proceudre writestr(c);
  begin
     if c<K then write(c);
     else begin
        writestr(left(T,c)); writestr(right(T,c));
     end;
  end;
function readtree;
  begin
     t:=readtreecode;  ( from F2 or F )
     if t=0 then begin
        c:=readcode;   ( from F1 or F )
        writestr(c);
        return(c);
     end else begin
        cl:=readtree; cr:=readtree; Cmax:=Cmax + 1;

        enter(T,cl,cr,cl cr,Cmax);
        return(Cmax);
     end;
  end;
procedure decode;
  begin
     Cmax:=K - 1;
     while not end of file do dummy:=readtree;
  end;
```

図2．復号化アルゴリズム

4．　評価

ソースプログラム、英文、実行プログラムについて、増分分解法、ＬＺＷ法と比較した結果を表１．に示す。

ＬＺＷ法に比べソースファイル等で約２割程度以上、英文と実行ファイルで１割以上圧縮率が向上している。繰り返す内容をひとまとめにして登録し、現れない記号列の登録をなくした効果が現れているといえる。

5．あとがき

以上、被圧縮データを隣接頻度によって解析し、切り出しする方法について述べた。

今後、最後までしている先読みを小量又はゼロにできる適応的なアルゴリズムへの変形をはかりたい。また、処理速度の改善も必要である。

参考文献

（１）J.Ziv and A.Lempel:"A universal algorithm for sequential data compression", IEEE Information Theory,23,3,pp.337-343(1977).

（２）J.Ziv and A.Lempel:"Compression of individual sequences via variable-rate coding",IEEE Information Theory,24,5,pp.530-536(1978).

（３）T.A.Welch:"Technique for high-performance data compression",Computer,17,6,pp.8-19(1984).

（４）朴　志煥,今井秀樹,山森丈範,伊藤秀一,石井正博：”パターンマッチングと算術符号を用いた実用的なデータ圧縮法”,信学論（Ａ）,Ｊ７１－Ａ,8,pp.1615-1624（昭63-08）.

（５）曽根和美,松嶋敏泰,鈴木　譲,平澤茂一：”ユニヴァーサルデータ圧縮法の実用化に関する研究”,情報理論とその応用学会,第１１回シンポジウム,pp.79-84（昭６３）

（６）横尾英俊：”Ｗｅｌｃｈ型データ圧縮法の新算法”,同上,pp.735-740.

Compression Ratio (and entropy assuming i.i.d.)

表１．実験結果

	増分分解法			ＬＺＷ法			本方法			圧縮率の改善	
ファイル名とサイズ[ﾊﾞｲﾄ]	圧縮率[%]	切出数[回]	登録数[回]	圧縮率[%]	切出数[回]	登録数[回]	圧縮率[%]	切出数[回]	登録数[回]	対増分分解法[%]	対ＬＺＷ法[%]
lzw.c	63.0	2887	2887	49.2	3727	3727	32.1+ 4.3=36.5	2895	979	42.1	25.8
10654	(31.5+22.9=54.4)		(1373)	(44.8)		(1721)	(28.1+ 3.6=31.7)		(678)		
parse.c	58.6	5096	5096	47.0	6803	6803	31.4+ 4.0=35.5	5055	1730	39.4	24.5
21084	(31.4+17.0=48.4)		(2300)	(42.6)		(3034)	(27.4+ 3.3=30.7)		(1209)		
lzw1.c	55.8	6350	6350	44.0	8089	8089	29.1+ 3.6=32.8	5976	2200	41.2	25.5
28029	(3.06+16.9=47.5)		(3064)	(40.8)		(3849)	(25.4+ 3.1=28.5)		(1460)		
ahuffman.pas	63.1	2197	2167	50.5	2849	2849	36.0+ 4.9=40.9	2287	731	35.2	18.0
7772	(32.0+19.4=51.3)		(976)	(44.5)		(1267)	(30.5+ 3.9=34.4)		(533)		
parse.pas	57.2	4417	4417	44.1	5525	5525	23.1+ 3.4=26.5	3281	1675	53.7	39.9
18481	(31.3+16.7=48.1)		(2293)	(40.7)		(2828)	(19.7+ 3.1=22.8)		(855)		
this.pas	51.0	6400	6400	39.2	7971	7971	25.0+ 3.1=28.2	5728	2025	44.7	28.1
30950	(28.5+14.2=42.6)		(3259)	(36.6)		(3979)	(21.6+ 2.6=24.2)		(1364)		
chkdsk.exe	78.4	3459	3459	73.6	5216	5216	57.2+ 7.0=64.2	4773	1030	18.1	12.8
10384	(36.5+30.7=67.2)		(1063)	(60.5)		(1393)	(51.5+ 4.7=56.2)		(775)		
diskcopy.exe	76.9	6132	6132	72.0	9102	9102	54.6+ 6.4=61.0	8021	1963	20.7	15.3
19604	(38.2+28.8=67.0)		(2064)	(60.6)		(2831)	(48.3+ 4.6=52.9)		(1404)		
parse.exe	86.3	10252	10252	82.4	15316	15316	62.9+ 6.8=69.7	13509	3075	19.2	15.4
30304	(44.0+31.5=75.5)		(3327)	(69.0)		(4670)	(54.7+ 4.7=59.4)		(2308)		
ps.man	57.0	2857	2857	45.2	3744	3744	33.8+ 4.3=38.1	3075	943	33.2	15.7
11648	(29.4+15.7=45.0)		(1235)	(40.5)		(1599)	(28.3+ 3.4=31.7)		(714)		
bs.man	54.7	4792	4792	43.4	6157	6157	33.3+ 3.9=37.3	5323	1332	28.5	14.1
21120	(29.2+14.7=43.8)		(2101)	(39.2)		(2867)	(28.8+ 2.8=31.6)		(1170)		
sh.man	52.2	6603	6603	40.4	8249	8249	27.9+ 3.3=31.3	6522	1779	40.0	21.8
31232	(28.8+13.4=42.2)		(3097)	(37.4)		(3878)	(24.8+ 2.5=27.3)		(1485)		

(注１）圧縮率欄の（）内は理想圧縮率である。増分分解法の左辺は登録符号の順序数と１記号の順序数とを、本方法の左辺はF_1とF_2とを別々に算出した値である。

(注２）登録数で（）内は、１回以上符号生成で使用した登録符号の数で内数である。