順序保存符号とその情報検索への応用

中津 楢男†

大容量半導体ディスクが実用化され，それを利用してデータベースが構築されることも夢ではないと考えられる．その場合にも記憶コストを考慮すればデータ圧縮は必要である．一般に，符号化されたデータを利用する場合には復号操作が必要になるが，その時間はファイルのアクセス時間に比べて小さいため，これまでは問題視されなかった．しかし半導体ディスク上に置かれた圧縮ファイルを扱う場合には，ファイルのアクセス回数とともにデータ復号時間が検索時間に大きな影響を及ぼす．本稿では復号操作なしにレコード検索が実現できる順序保存符号を提案する．まず Nakatsu, 1991 の結果を拡張し，r 元順序保存符号が存在するための必要十分条件を与え，最適 r 元順序保存符号の平均符号語長とハフマン符号の平均符号語長を理論的に比較する．次に2元符号化について，データ圧縮しない場合・ハフマン符号化・最適順序保存符号化を行った場合のデータ量，検索時間の実験結果を固定長ファイルの場合，可変長ファイルの場合についてそれぞれ報告する．この結果順序保存符号について，圧縮率はハフマン符号と同程度で約60%を達成し，検索時間はハフマン符号化を行った場合の 1/5（データ圧縮しない場合と同程度）であるということがわかり，検索対象となる属性の符号としてきわめて有効な符号であることが確かめられた．

An Alphabetic Code and Its Application to Information Retrieval

NARAO NAKATSU†

An alphabetic code is a code in which the numerical order of the codewords corresponds to the predefined order (for example, the alphabetical order) of the encoded symbols. Many information search techniques, such as the binary search method or the index tree search method, rely on the string comparison operations. If a record is coded using an alphabetic code, the comparison operation can be applied to the sequence of codewords, which saves time otherwise spent in decoding. This also saves storage space, in comparison with the case where no data compression method is applied. In this paper, a necessary and sufficient condition for the existence of an alphabetic code is presented. A bound of the redundancy of the optimal alphabetic code is also given in comparison with the Huffman code, which is tighter than those previously known. Experimental results show that the alphabetic code attains so good compression ratio as the Huffman code and it is five times faster than the Huffman code in in-core retrieval.

1. はじめに

データベースなど大量のデータを記憶する場合には，何らかの方法でデータ圧縮が行われる．データ圧縮によって実質的な記憶コストは低下するが，その代償として圧縮されたデータを利用する場合には，普通は元の記号に復元する必要があり，そのための復号時間が余分にかかる．しかし符号化を工夫すれば復号操作が不要になる場合がある（もちろん最終結果の表示には復号が必要である）．例えばデータベースで用いられる索引ファイル（B木[3]など）では，必要な操作はキーと呼ばれる文字列同士の比較だけであり，元の記号のアルファベット順が保存される符号化を用いれば符号語同士の比較によって求めるレコードへ到達できる．

低速の二次記憶装置を用いてデータベースを構築する場合には，こうした内部処理の高速化は無視できる程度の改良にしかならない．実際データベース操作に必要な時間の大半は二次記憶へのアクセス時間であり，従来の研究ではアクセス回数が効率の判断基準であった．しかし近年半導体記憶の集積度の向上は著しく，大容量半導体ディスクの出現によって1つのファイル全体が半導体ディスク上に置かれることも夢ではない．その場合にはファイルのアクセス回数と共に，データ復号時間が検索時間のなかで大きな割合を占めるものと考えられる．本稿では半導体ディスク上でのデータ検索を想定して，検索時間の低下を招くこと

† 愛知教育大学総合科学課程情報科学コース
Department of Computer and Information Sciences, Aichi University of Education

なくデータ圧縮を実現する順序保存符号について述べる．

静的符号化としてはハフマン符号が最適であることが知られており[10]，英文の場合では50%以上の圧縮率が達成されるといわれている[4]．ソースシンボルの持つ順序を保存するという条件を課した場合，平均符号長がどの程度長くなるかは興味ある問題である．

Gilbert と Moore は順序保存符号化のアルゴリズムを示した[1]．彼らのアルゴリズムによれば $H(S)+1 \leqq L < H(S)+2$ を満足する平均符号長 L をもつ符号が得られることが証明されている．(ここで $H(S)$ は与えられた情報源のエントロピーである)．最適な順序保存符号を計算する効率よいアルゴリズムは Hu らによって与えられているが，平均符号長については言及されていない[2]．二元の順序保存符号の平均符号長の上界は文献5)，6)で議論されている．文献6)では最初と最後の記号には $\lceil -\log p_i \rceil$，それ以外の記号には $\lceil -\log p_i \rceil + 1$ なる長さをもつ順序保存符号が存在することを示して上界を与えている．この符号の平均符号長は $L_{SF} + \Sigma p_i$ $(i=2, \cdots, n-1)$ である．ここで L_{SF} はシャノン・ファノ符号の平均符号長である．一方文献5)で示された上界を与える符号の平均符号長は $L_H + \Sigma p_i$ $(i=2, \cdots, n-1)$ より必ず小さい（文献5)の定義5，系2から)．ここで L_H はハフマン符号の平均符号長である．一般に $L_H \leqq L_{SF}$ なので文献5)の上界のほうが厳しい．

本稿では一般に r 元の場合について，順序保存符号が存在するための必要十分条件を与え，最適順序保存符号がハフマン符号に比べどの程度冗長かを明らかにする．次に順序保存符号のデータ検索への応用について述べ，最後に英単語辞書をデータとし，各種符号化を行った場合のデータ量と検索時間の実験結果を示す．

算術符号[4]は順序保存符号を生成し，漸近的最適であることが知られている．算術符号はファイル全体を圧縮するような場合には有用であるが，本稿で想定するような有限長レコードを各々符号化するような場合には適さない．それは一般に算術符号は極小符号とならないからである．例えば $S=\{a, b, c\}$ で生起確率を各々．5,．1,．4 とすれば，"bb" の符号長は算術符号では $\lceil \log 100 \rceil = 7$ となるが，実際は4ビットで十分である．このような理由から符号化の実験では算術符号は除外した．

2. 準　　備

以下本稿では情報源を S とし，符号 c としては瞬時に復号可能な符号（いわゆる語頭条件[10]（どの符号語も他の符号語の語頭にならない）を満足する符号）だけを考える．簡単のため符号アルファベットは r 元 $\{0, 1, \cdots, r-1\}$ とする．

[定義1] ソースシンボル x の符号語を $c(x)$ とする．ソースアルファベット上の順序を $<$，符号アルファベット上の順序を $\ll$ とする．$<$ も $\ll$ もそれぞれのアルファベット上で定義される記号列に対して，自然な形で拡大して定義されるものとする．$\forall x$, $\forall y \in S$, $x < y \Leftrightarrow c(x) \ll c(y)$ となる符号 c を順序保存符号☆(alphabetic code，以下 ap 符号と略す)という．以下では S の要素は $<$ に従って並び替えされているものとする．

[定義2] S と各シンボルの生起確率が与えられたとき，$L=\Sigma l_i p_i$ を平均符号長といい，L が最小となる ap 符号を最適 ap 符号という．ここで p_i はソースシンボル s_i の生起確率，l_i は s_i に割り当てされた符号語の長さである．最適 ap 符号の平均符号長を特に L_{op} とかく．

[例1] $S=\{s_1, s_2, s_3, s_4, s_5, s_6, s_7\}$，各々のシンボルの生起確率を 0.06, 0.15, 0.22, 0.08, 0.2, 0.05, 0.24 とする．3元ハフマン符号と3元 ap 符号の例を**図1**に示す．平均符号長はハフマン符号の場合は1.73，ap 符号の場合は1.76である．この場合の ap 符号は最適 ap 符号になっている．

3. ap 符号の存在条件

瞬時に復号可能な符号が存在する必要十分条件として Kraft の不等式が有名である[10]．本稿でもこれに倣い，ap 符号が存在するために符号語長が満足すべき必要十分条件を与える．

[定義3] ソースシンボル集合 $S=\{s_1, s_2, \cdots, s_n\}$ の各シンボルに割当された ap 符号の長さを $l_1, l_2, \cdots, l_n$ と

記号	確率	ハフマン符号	ap符号
s_1	0.06	011	00
s_2	0.15	02	01
s_3	0.22	2	02
s_4	0.08	010	10
s_5	0.2	00	11
s_6	0.05	012	12
s_7	0.24	1	2

図1 3元ハフマン符号と ap 符号の例
Fig. 1 A ternary Huffman code and an ap code.

☆ アルファベティック符号と訳すべきかもしれないが，情報検索への応用を意識して本稿ではこの訳を用いる．

する．$\alpha_i=\mathrm{Min}(l_{i-1}, l_i)$ $(2\leqq i\leqq n)$ とする．r進数xに対し，xの小数点以下y桁までを取り出す（つまり$y+1$桁以下を切り捨てる）関数を $\mathrm{Trunc}_r(y, x)$ とかく．xが小数点以下y桁未満の桁数しか持たないときは $\mathrm{Trunc}_r(y, x)=x$ である．

［定義4］ Sのi番目のシンボルについて，次の関数を再帰的に定義する．

$$\mathrm{Sum}(i)=\mathrm{Trunc}_r(\alpha_i, \mathrm{Sum}(i-1))+(1/r)^{\alpha_i} \quad (i\geqq 2)$$
$$=0 \quad (i=1)$$

このとき r 元 ap 符号について次の定理が成り立つ．特に $r=2$ の場合の証明は文献5)にある．

［定理1］ ap 符号の存在定理

ソースシンボル集合 S $(s_1<s_2<\cdots<s_n)$ の各シンボルに割当された符号語長を $l_1, l_2, \cdots, l_n$ とする．このような r 元 ap 符号が存在するための必要十分条件は $\mathrm{Sum}(n)<1$ である．

(証明) 付録に示す．

［例2］ $(l_1, l_2, l_3, l_4, l_5, l_6, l_7, l_8)=(2, 2, 2, 1, 2, 2, 3, 3)$ について3元 ap 符号が存在するかどうかを調べる．$(\alpha_2, \alpha_3, \cdots, \alpha_8)=(2, 2, 1, 1, 2, 2, 3)$ である．

$\mathrm{Sum}(1)=0$, $\mathrm{Sum}(2)=(1/3)^{\alpha_2}=1/9$, $\mathrm{Sum}(3)=\mathrm{Sum}(2)+(1/3)^{\alpha_3}=1/9+1/9=2/9$, $\mathrm{Sum}(4)=\mathrm{Trunc}_3(\alpha_4, 2/9)+(1/3)^{\alpha_4}=0+1/3=1/3$, $\mathrm{Sum}(5)=\mathrm{Trunc}_3(\alpha_5, \mathrm{Sum}(4))+(1/3)^{\alpha_5}=1/3+1/3=2/3$, $\mathrm{Sum}(6)=\mathrm{Trunc}_3(\alpha_6, 2/3)+(1/3)^{\alpha_6}=7/9$, $\mathrm{Sum}(7)=\mathrm{Trunc}_3(\alpha_7, 7/9)+(1/3)^{\alpha_7}=8/9$, $\mathrm{Sum}(8)=\mathrm{Trunc}_3(\alpha_8, 8/9)+(1/3)^{\alpha_8}=25/27<1$

であるので3元 ap 符号は存在する．実際，付録の定理1の証明にあるように，$c(s_i)$ を $\mathrm{Sum}(i)$ の3進表現のうち，小数点以下 l_i 桁と決めれば，$c(s_1)=00$, $c(s_2)=01$, $c(s_3)=02$, $c(s_4)=1$, $c(s_5)=20$, $c(s_6)=21$, $c(s_7)=220$, $c(s_8)=221$ が得られ，確かにこの符号は3元 ap 符号である．

4. ap 符号の平均符号長の上界

［定義5］ 与えられた情報源Sに対して，ハフマン符号化による符号長の割当を $l_1, l_2, \cdots, l_n$ とする．ただし $s_1<s_2<\cdots<s_n$ とする．$l_{i-1}>l_i$ かつ $l_i<l_{i+1}$ のとき i を第1極小点という．また $l_{i-1}>l_i=l_{i+1}=\cdots=l_{i+k}<l_{i+k+1}$ の時は i または $i+k-1$ の一方を第2極小点と呼ぶ．第2極小点は $p_i+p_{i+1}\leqq p_{i+k-1}+p_{i+k}$ の場合は i, そうでなければ $i+k-1$ を選ぶ．第1極小点の集合を D_1, 第2極小点の集合を D_2 と書き，極小点集合を $D(=D_1\cup D_2)$ とおく．ただし1とnは極小点には含めない．

［定義6］ Sのハフマン符号化によって得られる極小点集合を $D=\{i1, i2, \cdots, ik\}$ とする．次の手続きによって得られる D の部分集合 E を真極小点集合と言う．

1) $E\leftarrow\Phi$ (空集合)，$m\leftarrow 0$.

2) D が空なら終了．そうでなければ $k=|D|$ として，D の要素を $l_{i1}=l_{i2}=\cdots l_{ij}<l_{ij+1}\leqq\cdots\leqq l_{ik}$ となるように番号のつけ替えを行う．

3) $m\leftarrow m+1$, $E_m\leftarrow\{i1, i2, \cdots, ij\}$ とする．

4) $E\leftarrow E\cup E_m$. $D-E_m$(差集合)$=\{ij+1, \cdots, ik\}$ の中で対応する確率の大きい順に $(r-2)j$ 個の要素よりなる集合を F_m とする．$D-E_m$ の要素数が $(r-2)j$ より少ない場合は $D-E_m$ を F_m として終了．そうでなければ $D-E_m-F_m$ を改めてDとして2)へ行く．

定義6より極小点集合Dは $\bigcup_{t=1}^{m}(E_t\cup F_t)$ と等しい．

［例3］ 例1の情報源について考える．この場合 $(l_1, l_2, \cdots, l_7)=(3, 2, 1, 3, 2, 3, 1)$ なので，この場合3と5が第1極小点になる．また定義6によれば $r=3$ なので，$E_1=\{3\}$, $F_1=\{5\}$ で真極小点集合は $\{3\}$ である．

r元最適 ap 符号の平均符号長 L_{op} に関して，次の定理が示せる．ただしr元ハフマン符号化によって得られる平均符号長を L_H とする．

［定理2］ L_{op} の上界

与えられた任意の確率分布のもとで

$L_H\leqq L_{op}\leqq L_H+\Sigma p_i$ $(i\in E)+\Sigma p_{i+1}$ $(i\in E\cap D_2)$ が成り立つ．ここでEは定義6の真極小点集合である．

(証明) ap 符号はハフマン符号化を制限した符号となっているため $L_H\leqq L_{op}$ は自明である．$(s_1, s_2, \cdots, s_n)$ の生起確率を $(p_1, p_2, \cdots, p_n)$ として，ハフマン符号化による符号語長の割当を $(l_1, l_2, \cdots, l_n)$ とする．$L_H=\Sigma p_i l_i$ である．ここで $\forall s_k$ $(k\in E\cap D_1)$ については l_k+1, $\forall s_k, s_{k+1}$ $(k\in E\cap D_2)$ についてはそれぞれ l_k+1, それ以外の s_k には l_k という符号長の割当を行う．この結果得られる符号長を l'_k $(k=1, \cdots, n)$ とし，$\beta_i=\mathrm{Min}(l'_{i-1}, l'_i)$ とおく．こうした符号語長の組 $(l'_1, l'_2, \cdots, l'_n)$ は定理1を満たすことを示す．

$$\mathrm{Sum}(n)=\mathrm{Trunc}_r(\beta_n, \mathrm{Sum}(n-1))+(1/r)^{\beta_n}\leqq\mathrm{Sum}(n-1)+(1/r)^{\beta_n}\leqq(1/r)^{\beta_2}+(1/r)^{\beta_3}+\cdots+(1/r)^{\beta_n}$$

が成り立つ．この最右辺の式を(1)とする．l', β の定義から $k\in E$ ならば $\beta_k=\beta_{k+1}-l'_k=l_k+1$, $k\in D-E$ ならば $\beta_k=\beta_{k+1}=l'_k=l_k$ となり，l'_k $(k\in D)$ は(1)に2回寄与する．$k\notin D$ の時，l'_k $(\geqq l_k)$ は高々1

回寄与するだけである．しかも式(1)は $n-1$ 点の和であるため，一度も寄与しない l'_k も必ずある．したがって

$$\text{式}(1) < \sum_{k\notin D}(1/r)^{l'_k}+2\sum_{k\in D}(1/r)^{l'_k}$$

$$\leqq \sum_{k\notin D}(1/r)^{l_k}+2\sum_{t=1}^{m}\left(\sum_{k\in F_t}(1/r)^{l_k}+\sum_{k\in E_t}(1/r)^{l_k+1}\right)$$

$(D=\cup(E_t\cup F_t),\ E=\cup E_t,\ D-E=\cup F_t$ なので)

$$= \sum_{k\notin D}(1/r)^{l_k}+\sum_{t=1}^{m}\left(2\sum_{k\in F_t}(1/r)^{l_k}+2/r\sum_{k\in E_t}(1/r)^{l_k}\right)$$

$$= \sum_{k=1}^{n}(1/r)^{l_k}+\sum_{t=1}^{m}\left(\sum_{k\in F_t}(1/r)^{l_k}+(2/r-1)\sum_{k\in E_t}(1/r)^{l_k}\right) \qquad (2)$$

(なぜなら $\sum_{t-1}\left(\sum_{k\in F_t}(1/r)^{l_k}+\sum_{k\in E_t}(1/r)^{l_k}\right)=\sum_{k\in D}(1/r)^{l_k}$ かつ $\sum_{k\notin D}(1/r)^{l_k}+\sum_{k\notin D}(1/r)^{l_k}=\sum(1/r)^{l_k}$ なので)

さて E_t の要素数を j_t とする．E_t の求め方より E_t に属する番号に対応するシンボルはすべて同じ符号長を持つのでその長さを l_t とする．F_t の求め方から F_t の要素数は高々 $(r-2)j_t$ であり，しかも F_t に属する番号に対応するシンボルはすべて l_t+1 以上の符号長を持つ．したがって

$$\text{式}(2) < \sum_{k=1}^{n}(1/r)^{l_k}+\sum_{t=1}((r-2)j_t(1/r)^{l_t+1}$$

$$+(2/r-1)j_t(1/r)^{l_t})=\sum(1/r)^{l_k}\leqq 1$$

(なぜなら $\{l_k\}$ はハフマン符号の符号長でクラフトの不等式[10]を満たすから)．定理1より l'_k $(k=1,\cdots,n)$ という符号長の割当をもつ ap 符号が存在することが言えた．この ap 符号の平均符号長は $L_H+\sum p_i$ $(i\in E\cap D_1)+\sum p_i+p_{i+1}$ $(i\in E\cap D_2)$ で，$D_1\cap D_2=\Phi$ から，L_{op} の上界として定理2が得られる． ■

[例4] 例1の情報源について考える．$D=\{3,5\}$，$E=\{3\}$ である．$p_3=.22$ なので，確かに $1.76=L_{op}<L_H+p_3=1.95$ が満足されている．

定理2より L_{op} は L_H よりごくわずか $(\sum p_i$ $(i\in E)+\sum p_{i+1}$ $(i\in E\cap D_2))$ 長くなるだけであることが証明できた．実際どの程度長くなるかについては6章で述べる．

5. ap 符号の応用

情報検索においては2つのキーの値の比較が本質的な操作であると考えられる．例えばソート済みファイルの2分探索や2次索引（探索木で実現することが多い）を用いた検索では，いずれの場合も与えられた値とデータベースに記憶された値の比較が必要とされる．また直接アクセスだけでなく順アクセスの必要性も高い．このような場合検索属性が ap 符号を用いて符号化されておれば，符号語のままの比較操作で検索が可能である．5，6章では2元符号に話を限定する．

例えば**図2**の符号表に従って符号化したファイルの一部を**図3**に示す（キー以外の内容は省略．各レコードはキーの昇順に並べてある）．ap 符号化では符号化後も各レコードがキーの昇順に並んでいることに注目すべきである（ただし符号語同士の比較は左づめの論理比較とする）．範囲検索でキー値が "bad" から "bd" の範囲にある単語を求める場合，ap 符号では "bad¥" を符号化した "100111100" と "bd¥" を符号化した "1011100" の間のレコードすべてが該当するレコードであることがわかる．しかしながらキー値がハフマン符号で符号化されている場合には順序保存の性質がないため，まず復号を行ってからでないとキー値の比較ができない．このように ap 符号では符号語のままの比較で値の比較が実現できる．実際の応用では検索方法に応じて ap 符号にいくつかの工夫をこらす必要が生じる．

つぎに固定長レコードと可変長レコードにわけて，ap 符号化の問題を考える．

5.1 固定長レコード用への応用

1つのレコードは通常複数のフィールド（属性）よりなるが，この場合はレコード長も各フィールドも固定長として実現される．キー属性の定義域が文字列の場合，このキー属性に対応するフィールドの大きさは考えられ得る値の最長の文字列が納まるよう確保する

記号	ハフマン符号	ap符号
¥	10	00
a	1110	01
b	0	10
c	110	110
d	1111	111

図2 符号化の例
Fig. 2 Examples of codes.

	もとのファイル	ハフマン符号化	ap符号化
レコード1	ba¥	0111010	100100
レコード2	bac¥	0111011010	100111000
・	bb¥	0010	101000
・	bc¥	011010	1011000
レコード5	c¥	11010	11000

図3 符号化前のファイルと図2に従って符号化した後のファイルの例
Fig. 3 An original file and the files coded by Fig. 2.

のが普通である．キー属性を ap 符号化する場合の最適データ圧縮としては符号化後の長さの最大値が最小となるような符号を見つける必要がある．

[例5] 属性値集合 {*aab, aba, acb, acbc, baa, bab*} を固定長でデータ圧縮する．平均符号長を最小にする最適 ap 符号は $\{a:0, b:10, c:11\}$ であるが（平均符号長＝1.53 ビット），この符号化では最も長い属性値は *acbc* で 0111011 のように 7 ビットに符号化される（終端記号は考慮しない）．したがってこれらを固定長で表現しようとすればフィールド幅として 7 ビット必要となる．しかし符号 $\{a:00, b:01, c:1\}$ によれば 6 ビットあれば各属性値を符号化できる（平均符号長＝1.84 ビット）．

この問題はソースシンボル s_j $(1 \leqq j \leqq n)$ の符号語長を l_j，属性値 x_i に現れる記号 s_j の出現回数を c_{ij} としたとき $L_i = \Sigma c_{ij} l_j$ の最大値 $(1 \leqq i \leqq N$, N は全レコード数）を最小とする l_j の割当を求める問題であり，N 個の n 元1次式の上方エンベロープを求める問題に関係し，一般には難しい問題である[8]．

ap 符号の可能な符号木の数 $f(n)$（n はシンボル数）は $f(n) = \Sigma f(i) f(n-i)$ ただし $f(1)=1$, で与えられるがこのすべてを考慮すれば指数時間かかる．そこで，長さの長い属性値だけを対象に符号を決め，それで不都合なければ終了するという楽観的なアルゴリズムを用いることにした．具体的なアルゴリズムは文献7)で与えられているがここでは紙面の都合上省略する．いったん符号化が終わればキー属性に対応したフィールドは固定長なのでそのフィールドの値（ap 符号で符号化されている）を使って検索できる．このアルゴリズムを用いた実験結果は 6.1 節で与えられている．

5.2 可変長レコードへの応用

可変長レコードでは通常フィールド長も可変である．各フィールドを同定するために，レコードの先頭に各フィールドの開始位置を示すディレクトリをもうけたり，フィールドごとにデリミタ（区切り記号）を用いたりすることが多い．ディレクトリを用いる場合には各フィールドの開始位置がディレクトリを見ることでわかるので，最適 ap 符号がそのまま利用できる．

フィールドがデリミタで区切られている場合や UNIX システムのようにファイルがストリーム（ファイルは文字の並びでレコードは改行符号で区切られている）の場合には，符号化に工夫をしなければ区切りが認識できない場合がある．もちろん先頭から復号を行えば区切りが正しく認識できるのであるが，そうすれば復号せずに符号語間の比較を行うという ap 符号の特徴が消えてしまう．このことを例で説明する．

UNIX の英単語辞書は単語の区切りをデリミタ（改行符号）で表現している．ASCII 符号は固定長符号なので，ファイルの中で改行符号 0d が現れればそれが単語の切れ目とわかる．しかし ap 符号化やハフマン符号化ではデリミタも1つのシンボルとして符号化される．これらは可変長符号であるため，デリミタと同じビット列が符号語の連接の部分列として現れる場合がある．こうしたところをデリミタと誤って認識すると，単語の切れ目を誤る．例えば図2の ap 符号では，“aca¥” は “011100100” と符号化されるが5，6番目のビット列をデリミタ¥と見なすかもしれない．

符号語の任意の連接の部分列としてデリミタが現れなければ，ファイルのどの部分から調べ始めてもデリミタの存在が認識できる．このような性質を持つ符号をデリミタ用 ap 符号という．順序保存であるためにはデリミタの符号語はすべて0より成っている必要がある．例えば図2において，b，c の符号語をそれぞれ 101，1101 とすればデリミタ用 ap 符号が得られる（¥以外のシンボルの符号語をどのように組み合わせても 00 が現れないから）．

平均符号長が最小なデリミタ用 ap 符号を考える．デリミタ長を dell，各シンボルの符号語の接頭語として現れる0の連の最大長を l（ただし $0 \leqq l <$ dell）としたとき，この条件を満たす最適デリミタ用 ap 符号（これを（dell, l）ap 符号という）はデリミタの頻度および各ソースシンボルの頻度に応じて一意に決まる．(dell, l) ap 符号の各符号語の接頭語としては高々長さ l の0の連が許され，また接頭辞として許される0の連の長さも高々 dell$-l-1$ である（デリミタ以外の符号をどのように組み合わせてもデリミタにならないようにするため）．最適な (dell, l) ap 符号を計算するアルゴリズムは文献7)で与えられているが，ここでは割愛する．

6. 実験結果

ap 符号の有効性を調べるため，実際のファイルに対して ap 符号化を適用し，その圧縮率，検索時間を調べた．実験用ファイルとしては HP Domain ワークステーションの英単語辞書（/sys/dict）を利用した．この辞書の各文字の出現頻度を数え，その頻度をもとにハフマン符号化，最適 ap 符号化を行った．この辞書データに対する符号語の割当結果の一部を**図4**に示

文字	頻度	ハフマン	最適ap	(4,2)ap符号
区切り	42695	101	000	0000
a	26385	0011	00101	001001
b	6355	011100	001100	00101
c	13770	00100	001101	00110
d	13214	00101	00111	00111
e	38017	111	010	010
f	4690	110001	011000	0110001
・・・	・・・	・・・	・・・	・・・・・
計	374709	L=4.26	L=4.43	L=4.64

図 4 辞書ファイルに出現する文字の頻度とその符号語の一部

Fig. 4 The frequency and corresponding codewords of some symbols for the dictionary file.

す.

符号化によるファイルの大きさの理論値は以下の通りである（単位はバイト）．ハフマン符号，ap 符号とも可変長符号であるため符号化後のファイルはビット列として大きさを計算した．また区切り記号は頻度を 42695（単語数）として符号化した．

ASCII 符号	ハフマン符号	最適 ap 符号
374709	199955	204311

6.1 固定長レコード

単語長の最大値は 22 であったのでこの辞書を固定長レコードの集まりと表現するには各単語あたり 22 バイトを必要とする．一方 5.1 節で示した方法を用いて/sys/dict を固定長レコード用 ap 符号で表現した場合の符号化後の単語の最大長は 101 ビットとなった．この場合バイト境界を考慮しても 13 バイトで各単語を表現できる．この符号は長さ 21 以上の単語集合に対する最適 ap 符号であり，この符号化によってどの単語も 101 ビット以下で符号化できることを確認した．

固定長レコードでは検索は固定長キーの比較によってなされるため両方の符号化で検索時間の差はないものと考えられる．したがって固定長ファイル用の符号の場合は ap 符号を用いることで検索時間の低下を招くことなく 41% のデータ圧縮が可能となる．

6.2 可変長レコードの場合

/sys/dict ファイルは各レコード（単語）がデリミタで区切られているので 5.2 節で議論したようにデリミタ用 ap 符号とデリミタ用ハフマン符号を用いて実験した．

実験：/sys/dict ファイルを最適なデリミタ用 ap 符号で符号化したファイル apdict およびデリミタ用ハフマン符号で符号化した dhdict を作成し，圧縮しないままの辞書を含めて 3 つのファイルの大きさと，各々を 2 分探索で検索した場合の平均検索時間を測定した．デリミタ用 ap 符号の中では (4,2) ap 符号が最も圧縮効率が良かった（図 4）．

また通常の符号化では単語が 1 バイトの途中から始まる場合も生じ得るため，必ず単語はバイト境界で終了するよう必要なだけ 0 を補って符号化した（こうしても順序が保存されることに注意）．例えば *babe* は 00101001・00100101・0100000 と符号化されるべきであるが（図 4 参照），00101001・00100101・01000000 のように符号化した．これはバイト単位でしかファイルにアクセスできないためである．(4,2) ap 符号の性質から 1 バイトを読み込みその下位 4 ビットがすべて 0 であればそこが単語の切れ目であることが認識できる．dhdict についても同様に符号化した．このように符号語の最後にいくつかの 0 を付け足す場合には平均符号長最小の符号が必ずしもファイルサイズを最小にするとは限らないが，この実験では平均符号長最小の符号を用いた．ハフマン符号についても平均符号長は同じでも，符号の選び方で最終的なファイルサイズは変動するが，この点も考慮しておらず，符号化には文献 9)のアルゴリズムを用いた．

この実験結果を次に示す．検索時間はいずれもランダムに選んだ 50 の英単語を検索するのに要した時間を示している．1 章で述べたようにここでは半導体ディスクでの検索を想定しているため，各ファイルは主記憶上に置き，その上で 2 分探索を行った．この時間には入出力のための時間は含まれていない．

	/sys/dict	dhdict	apdict
データ量（B）	374709	228492	230643
検索時間 (ms)	192.6	978.2	187.3

ハフマン符号と ap 符号を比較すれば，デリミタ用ハフマン符号とデリミタ用 ap 符号で圧縮率はほとんど変わらず (0.9% 増)，検索時間はデリミタ用 ap 符号のほうがはるかに速い（約 5 倍）．apdict のほうが /sys/dict より速いのは，apdict のほうがファイルの大きさが小さいためファイルアクセス回数が少なくて済むためと考えられる．

この結果より ap 符号はアクセス時間の低下を招くことなく（つまりデータ圧縮しない場合と同じ時間で），データ圧縮を実現することが確かめられた．

ちなみにファイルを磁気ディスクに置いた場合には ap 符号のほうがハフマン符号化した場合の検索時間より約 3 % 速くなるだけであった（ディスクアクセス時間がほとんどで復号のための時間差はそれに比べてごく小さいため）．

7. おわりに

本稿ではまず最適順序保存符号が存在するための必要十分条件を与え，得られた符号の平均符号長がハフマン符号の場合に比べそれほど悪くならないことを理論的に示した．

次に ap 符号を応用して実際のファイルのデータ圧縮を行い，その圧縮率とデータの検索時間を測定した．その結果 ap 符号の場合には復号操作を行わずに検索が可能であるため，検索時間は圧縮を行わない場合とほとんど変わらないことが確かめられた．データ量については原データのほぼ 60% となり，最適符号と言われるハフマン符号化と大差のない圧縮率が得られた．

ap 符号は符号化の段階でデータ圧縮を行うためいわゆるファイル構成には無関係で，任意のファイル構成に対して適用可能である．応用分野としてはキーの順序を用いて検索が行われる場合すべてが考えられる．例えば関係表の主キー属性の符号化，2分探索が行われるシーケンシャルファイル，辞書，探索木など比較だけが重要な分野に広く応用できると考えられる．日本語データへの応用は残された問題である．

謝辞 本研究の実験用プログラムの開発にご協力いただいた，秋山奈美さん（日立製作所，旭工場）と片桐友子さん（富士通 VLSI）に感謝いたします．また貴重なご示唆をいただいた査読者に深謝いたします．

参考文献

1) Gilbert, E. N. and Moore, E. F.: Variable Length Binary Encodings, *Bell Syst. Tech. J.*, Vol. 38, No. 4, pp. 933-967 (1959).
2) Hu, T. C. and Tucker, A. C.: Optimal Computer Search Trees and Variable-Length Alphabetic Codes, *SIAM J. Appl. Math.*, Vol. 21, No. 4, pp. 514-532 (1971).
3) Knuth, D. E.: *The Art of Computer Programming: Sorting and Searching*, Vol. 3, Addison-Wesley (1973).
4) Lelewer, D. A. and Hirschberg, D. S.: Data Compression, *ACM Comput. Surv.*, Vol. 19, No. 3, pp. 261-296 (1987).
5) Nakatsu, N.: Bounds on the Redundancy of Binary Alphabetic Codes, *IEEE Trans. Info. Theory*, Vol. 37, No. 4, pp. 1125-1129 (1991).
6) Yeung, R. W.: Alphabetic Codes Revisited, *IEEE Trans. Info. Theory*, Vol. 37, No. 3, pp. 564-572 (1991).
7) 秋山：順序保存符号の有効性の検証，愛知教育大学情報科学コース卒業研究報告書 (1990).
8) 今井：多次元 Davenport-Schinzel 列計算における線形化手法とその応用，情報処理学会研究報告，89-AL-9 (1989).
9) 大野(訳)：データ構造とアルゴリズム，培風館 (1987).
10) 宮川(訳)：情報理論入門，p. 229, 好学社 (1969).

付録 定理1の証明

順序保存符号は仮定より語頭条件を満たすので，符号木で一意に表現できる．r 元符号の符号木とは r 分木で次の性質を持つものをいう．(1)各節点より出る枝には 0 から $r-1$ のいずれかが重複なくラベル付けされている．(2)その木はちょうど n 個の葉を持ち，各葉はそれぞれソースシンボルの1つに対応している．そして符号木の根より s_1 に対する葉へ至る道上のラベルの連接が s_i の符号語に等しい．

s_i の符号語 $c(s_i)$ を r 進小数（小数点は $c(s_i)$ の左端にあると考える）と考えた場合，それが表す値を $v(c(s_i))$ と書く．例えば $r=3$ で，$c(s_i)=012$ の時，$v(c(s_i))=(1/3)^2+2(1/3)^3=5/27$ である．

このとき次の補題が成り立つ．

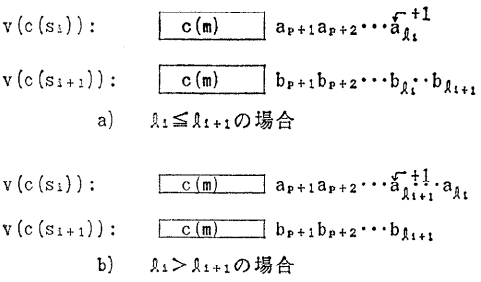
[補題] 任意の順序保存符号について，$\forall k, \mathrm{Sum}(k) \leqq v(c(s_k))$ が成り立つ．

(証明) $k=1$ の時，$\mathrm{Sum}(1)=0$ で $v(c(s_1)) \geqq 0$ なので補題は成り立つ．$k=i$ で補題が成り立つと仮定する．与えられた順序保存符号の符号木を考える．s_i と s_{i+1} に対応する葉節点の最小共通祖先を m とする．m の深さを p とする（根の深さは 0）．順序保存符号，および符号木の定義から $c(s_i)=c(m)a_{p+1}a_{p+2}\cdots a_{l_i}$, $c(s_{i+1})=c(m)b_{p+1}b_{p+2}\cdots b_{l_{i+1}}$（ただし，$c(m)$ は根より m に至る道上のラベルの連接でその長さを p とする．また，$a, b \in [0, r-1]$ で，$c(s_i) \ll c(s_{i+1})$ なので $a_{p+1}<b_{p+1}$ と書ける．$l_i \leqq l_{i+1}$ の時，$\alpha_{i+1}=l_i$ なので

$$\begin{aligned}
&\mathrm{Sum}(i+1)=\mathrm{Trunc}_r(\alpha_{i+1}, \mathrm{Sum}(i))+(1/r)^{\alpha_{i+1}}\\
&\leqq \mathrm{Trunc}_r(l_i, v(c(s_i))+(1/r)^{l_i}=v(c(s_i))+(1/r)^{l_i}\\
&\qquad (c(s_i)\text{ の長さは } l_i \text{ なので})\\
&=v(c(m))+\sum_{j=p+1}^{l_i} a_j(1/r)^j+(1/r)^{l_i}\\
&\leqq v(c(m))+(a_{p+1}+1)(1/r)^{p+1}\\
&\leqq v(c(m))+b_{p+1}(1/r)^{p+1} \leqq v(c(s_{i+1}))
\end{aligned}$$

直感的にいえば，$v(c(s_i))+(1/r)^{l_i}$ は $c(s_i)$ の最下位の桁に1を加えることを意味し，桁上がりを考慮してもその値が $v(c(m)(a_{p+1}+1))$ を越えない（**付図1a**）．$l_i > l_{i+1}$ の時，$\alpha_{i+1}=l_{i+1}$ なので

$$\mathrm{Sum}(i+1) \leqq \mathrm{Trunc}_r(l_{i+1}, v(c(s_i)))+(1/r)^{l_{i+1}}$$



付図 1 $\mathrm{Trunc}_r(v(c(s_i)),\alpha_i)+(1/r)^{\alpha_i}<v(c(s_{i+1}))$ の証明
App. fig. 1 The proof of $\mathrm{Trunc}_r(v(c(s_i)),\alpha_i)+(1/r)^{\alpha_i}<v(c(s_{i+1}))$.

$$=v(c(m))+\sum_{j=p+1}^{l_{i+1}}a_j(1/r)^j+(1/r)^{l_{i+1}}$$

$$\leqq v(c(m))+(a_{p+1}+1)(1/r)^{p+1}$$

$$\leqq v(c(m))+b_{p+1}(1/r)^{p+1}\leqq v(c(s_{i+1}))$$

$c(s_i)$ が l_{i+1} で切り捨てされる以外は上と同じ（付図1b参照）．よって補題は証明された． ■

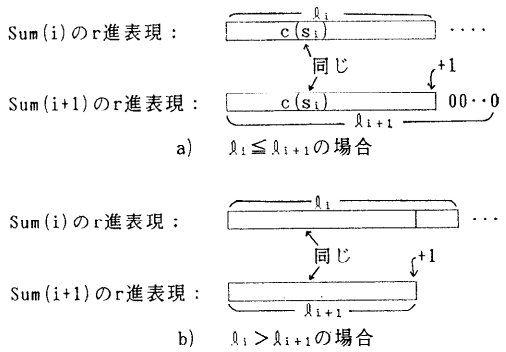
[定理1の証明]

任意の ap 符号について，v の定義から $v(c(s_i))$ は常に1より小さい．したがって補題を用いて $\mathrm{Sum}(n)\leqq v(c(s_n))<1$ がいえる．

逆に $\mathrm{Sum}(n)<1$ の時，指定された長さを持つ ap 符号を次のように求めることができる．

符号の作り方：$\mathrm{Sum}(i)$ を r 進小数で表現し，その小数点以下 l_i 桁を $c(s_i)$ とする．

こうして求めた符号が ap 符号であることを証明する．そのために任意の i について，$c(s_i)\ll c(s_{i+1})$ でしかも $c(s_i)$ が $c(s_{i+1})$ の語頭にならないことをいえば良い．$l_i\leqq l_{i+1}$ の時，$\alpha_{i+1}=l_i$ なので $\mathrm{Sum}(i+1)=\mathrm{Trunc}_r(l_i,\mathrm{Sum}(i))+(1/r)^{l_i}$．これは $\mathrm{Sum}(i+1)$ が $\mathrm{Sum}(i)$ を小数点以下 l_i 桁で切り捨てし，l_i 桁目に1を加えて得られることを意味している．したがって $\mathrm{Sum}(i+1)<1$ なので，$c(s_i)$ と $c(s_{i+1})$ は始めの l_i 桁のどこかで少なくとも一箇所違っており，$c(s_i)\ll c(s_{i+1})$ である（付図2a参照）．



付図 2 $c(s_i)$ が $c(s_{i+1})$ の語頭にならない証明
App. fig. 2 It is shown that $c(s_i)$ is not a prefix of $c(s_{i+1})$.

$l_i>l_{i+1}$ の場合 $\alpha_{i+1}=l_{i+1}$ なので $\mathrm{Sum}(i+1)=\mathrm{Trunc}_r(l_{i+1},\mathrm{Sum}(i))+(1/r)^{l_{i+1}}$．これは $\mathrm{Sum}(i+1)$ が $\mathrm{Sum}(i)$ を小数点以下 l_{i+1} 桁で切り捨てし，l_{i+1} 桁目に1を加えて得られることを意味している．したがって $\mathrm{Sum}(i+1)<1$ なので，$c(s_i)$ と $c(s_{i+1})$ は始めの l_{i+1} 桁のどこかで少なくとも一箇所違っており，$c(s_i)\ll c(s_{i+1})$ である（付図2b参照）．いずれの場合も $c(s_i)\ll c(s_{i+1})$ で $c(s_i)$ は $c(s_{i+1})$ の語頭でないことが証明されたので $\{c(s_i)\}$ は ap 符号である． ■

(平成3年 9 月 27 日受付)
(平成4年11月 12 日採録)

中津 楢男（正会員）

1953 年生．1977 年京都大学工学部情報工学科卒業．1979 年同大学院修士課程修了．同年愛知教育大学教育工学センター助手，現在，同大学総合科学課程情報科学コース助教授．工学博士．データベース，効率良いアルゴリズムの設計，記憶と学習に興味を持っている．電子情報通信学会，ACM 各会員．