ハフマンコード表の圧縮とその応用

河村知行† 江口賢和† 重村哲至†

ハフマンコードによるデータの圧縮と復元を行うためのハフマンコード表を小さくする方法を開発したので報告する．この方法により，従来方式に比べてハフマンコード表の大きさを約4分の1にすることができた．また，これを他のデータ圧縮法と併合することにより，UNIX の compress コマンドによる平均圧縮率45%を32%に大幅に改善できた．

Compression of Huffman Code Table and Applications

Tomoyuki Kawamura,† Yoshikazu Eguchi† and Tetsuji Shigemura†

We developed a method to compress Huffman code table, which is required in order to compress and decompress data with Huffman code. This method reduces Huffman code table to about 1/4 in comparison with the former method. We applied this to another compression method, and achieved very good compression ratio (32%) compared with that (45%) by "compress" command on UNIX.

1. はじめに

データ圧縮を行う際によく用いられる技法としてハフマンコード（以後 HC と略す）がある[1),2)]．HC は種々のデータの圧縮に適用可能である．そして，対象となるデータの基本要素（以後，文字と呼ぶ）の出現確率を固定のものと見なすことが基本となっている．

計算機内のファイルを対象データとすると，データごとに文字の出現確率が大きく異なることが多い．その場合は，「データ固有の HC 表」と「HC に変換したデータ」を合わせたものを圧縮データとすればよい．これにより，より良い圧縮率が得られる．しかし，データごとに HC 表を取り替えなければならず，一般に HC 表は大きいのでこの方法には限界があった．

われわれは，HC 表を小さな記憶量で表現する方法を開発したのでこれについて報告する．この方式により HC 表の大きさを，従来方式に比べて約1/4にできた．

また，本方式と他の方式[3)]を併合することで画期的な圧縮率を得たのでそれについても報告する．

† 徳山工業高等専門学校情報電子工学科
Department of Information and Electronics Engineering, Tokuyama College of Technology

2. 従来のハフマンコード表の表現

2.1 表現の例

データ固有の HC 表によるデータの圧縮例を**図1**に示す．HC のパターンはデータの出現回数により計算される．図1(a)の元データに対する HC は，図1(b)のようにして決定される．そして，HC 表は図1(c)のようになる．(c)と(a)の HC 表現を合わせたものが圧縮データであり，図1(d)のように表現できる．(d)の最初の256ビットは各文字（0〜255）がデータの中に存在するか否かの判定のための情報である．この情報と次の「長さ」と「パターン」の5回の繰り返しで HC 表を表現している．実用データでは，この繰り返しは100〜256回となる．「長さ」は5ビット固定長である．「パターン」はビット可変長である．(d)の最後の行が(a)を HC で変換したものである．データは

$$256+(5+2)\times 3+(5+3)\times 2+2\times 7+3\times 2$$

の313ビットで表現できたことになる．

2.2 パターンの長さの上限

この例では，「パターン」は2〜3ビット長であるが実用データでは1〜19ビット長となる．これが，HC 表を大きくする最大の原因である．しかし，実用データにおいてはパターンは31ビット以内に納まるので，「長さ」は5ビット固定長で十分である．その理由を説明するために，少ない出現文字数でパターン

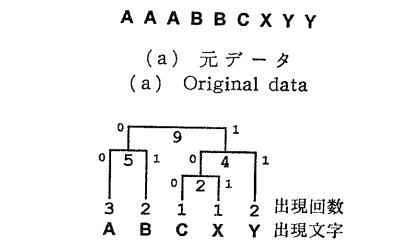

図 1 従来のハフマンコードの例
Fig. 1 An example of former Huffman code.

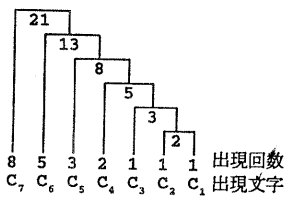

図 2 パターンを長くする最悪の場合
Fig. 2 The worst growth of pattern.

の長さが長くなってしまう最悪の場合を考えよう．その場合の文字の出現回数を**図 2** に示す．

図 2 は，図 1 (b) と同じく木の高さがパターンの長さに対応する．図からわかるように長さ（高さ）6 のパターンが発生するには最低でも 21 文字必要である．これを関数 f とすると $f(6)=21$ である．そして，図からわかるように $f(n)=\mathrm{fib}(n+1)$ である．fib はフィボナッチ関数である．よって $f(32)=5,702,887$ となり，パターンの長さが 32 になるためには最低でも 5,702,887 文字必要となる．これにより実用データにおいても「長さ」を 5 ビット固定長で表現しても問題はない．

3. ハフマンコード表を小さくする方法

以下に述べる HC 表を小さくする方法 CHT (Compact Huffman code Table) と呼ぶ．

3.1 CHT の原理

CHT の特徴を一言で言えば，「パターンを記憶する代わりに長さを記憶する」ということである．図 1 (d) の「長さ」だけを記憶すればよいので，「パターン」は必要なくなり，HC 表を大きくする最大の原因を取り除くことができる．

「長さ」だけ記憶すればよい理由は，長さからパターンを作ることができるからである．**図 3** はそのようすを示している．図 1 (b) では文字の出現回数が最も少ない 2 つを統合して，木を構成していく．統合箇所の下の数は，合計された出現回数である．図 3 では，パターンの長さにより統合を行う．長さが最も長い 2 つを統合して木を構成していく．統合箇所の下の数は，1 つ短くなった長さである．

CHT では，図 3 によりパターンを決定する．このパターンは，図 1 (b) による従来の HC のパターンと異なる．出現回数（確率）によりパターンを決定する従来の方法では 'A' が 'CX' と統合されることはありえない．このように，図 1 (b) と図 3 では各文字のパターンは異なりうる．しかし，パターンの長さは必ず同じになる．よって，2 つのパターン集合は同じ圧縮率をもたらす．このように，図 1 (b) の代わりに図 3 を用いることができるのは，HC が出現回数の情報をかなり端折った表現になっているからである．

3.2 処理過程

以上をまとめると，CHT による HC のパターンは次の (あ) と (い) により決定される．

(あ) 出現回数から，各文字ごとのパターンの長さを求める．

(い) 長さから，各文字ごとのパターンを求める．

そして，(あ) の情報と (い) を用いた各文字の変換結果

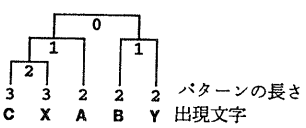

図 3 長さによるパターンの決定
Fig. 3 Decision of pattern from length.

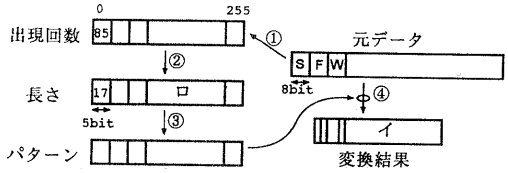

図 4 CHT の処理の基本
Fig. 4 Basic process of CHT.

を圧縮データとする.

一方，データの復元は，(い)と完全に同じアルゴリズムで(あ)の情報を処理して HC のパターンを得る．そのパターンで上述の変換結果を処理して各文字を復元する.

圧縮データを得る過程を図解すると**図 4** になる．①により各文字の出現回数を求める．②によりパターンの長さを求める．③によりパターンを求める．④によりパターンを用いて各文字を変換する．最後に，「長さ」: ロと「変換結果」: イを記憶する.

3.3 ハフマンコード表の圧縮

以上で CHT の基本概念の説明は終わりである．しかし，実際の CHT では「長さ」の情報を「元データ」と同じように HC で記憶する．それを**図 5** に示す．⑦⑧を行う前に，③④⑤⑥を行うことで「長さ」: ロを「変換結果」: ハで表現する．配列「ニ」は，その両端の連続する零要素を除いた要素だけで表現する．そして，最後に「ニ」「ハ」「イ」を圧縮データとして記憶する．これにより，図 4 より良い圧縮率を得ている.

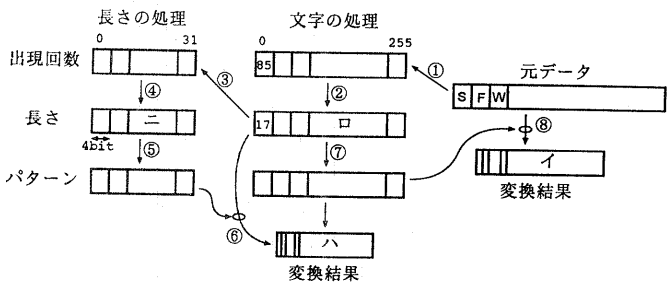

図 5 実際の CHT の処理
Fig. 5 Real process of CHT.

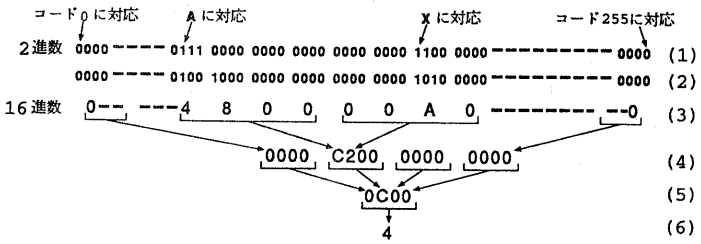

図 6 存在判定情報の圧縮
Fig. 6 Compression of decision data for existence.

「ニ」の各要素は 4 ビット固定長である．なぜなら，2.2 節で示したように $f(16)=2,584$ なのでパターンの長さが 16 になるためには最低でも 2,584 個の文字が必要である．しかし，図 5 からわかるように，文字(図 5 のロの要素)は 256 個 (配列の添え字が 0〜255) しかないので，パターンの長さは 15 以下である．よって，4 ビットあれば十分表現できる.

3.4 存在判定情報の圧縮

また，図 1 (d)の先頭の 256 ビットの存在判定情報は，**図 6** のように圧縮を行っている．図 6 (1)の隣あったビットの排他的論理和をとったものが(2)である．(2)の 16 進数表現が(3)である．(3)の 0 でない値を 1 で，0 を 0 で表して，4 ビットずつまとめたものが(4)である．これを再帰的に繰り返すことで(4)→(5)と(5)→(6)を行っている．最終的には「4 CC 482 A」の 28 ビットで 256 ビットの(1)を表現している.

復元も再帰的処理により行う．「4 CC 482 A」の先頭の 4 は 2 進数で 0100 あるから，この中の 3 つの 0 で 256 ビット中の先頭の 64 ビットと最後の 128 ビットがすべて 0 であることがわかる．1 に対応した C (「4 CC 482 A」の 4 の直後の C) に対して再帰的処理を行えば復元ができる.

3.5 ハフマンコード表の大きさ

実用データに対する，従来方式と CHT 方式による HC 表の大きさを**表 1** に示す，表の中の prg 1 prg 2 prg 3 は言語 C によるソースプログラム，roff は roff コマンドのデータ，tex は tex コマンドのデータ，man は日本語テキスト，68k は MC 68030用の実行形式，r3k は R 3000 用の実行形式である.

表 1 の従来方式①は，図 1 (d)の「存在判定情報」と「長さ」と「パターン」の総ビット数を表す．CHT 方式②は，「存在判定情報」と図 5 の「ニ」「ハ」の総ビット数を表す．「存在判定情報」は両方とも圧縮し

表 1 CHT によるハフマンコード表の圧縮
Table 1 Compression of Huffman code table by CHT.

		prg 1	prg 2	prg 3	roff	tex	man	68 k	r 3 k
大 き さ	(byte)	79088	12121	2091	105596	92648	75461	77824	102400
出現文字数	(個)	96	72	69	98	113	218	256	256
従来方式①	(bit)	1408	1060	971	1468	1796	3303	3898	3894
CHT 方式②	(bit)	464	414	366	453	531	881	867	839
圧縮率 ②/①	(%)	33	39	38	31	30	27	22	22

てある．②/①の値は，22〜39% とかなり小さくなっている．特にバイナリデータでの効果が顕著である．

3.6 文字コードの拡張

ここまで「文字」を値 0〜255 をもつ8ビットデータとして扱ってきた．しかし，HC はより多くの値をもつ「文字」に対しても有効である．その場合でも，CHT 処理は図5をそのまま実行できる．ただし，図5の「文字の処理」の配列の大きさは 0〜255 ではなく 0〜2047 などとなる．

4. CHT の応用

各文字の値が 255 を大きく超える場合に CHT は非常に有効である．なぜなら，HC 表は各文字ごとの長さとパターンを記憶する必要があり，その文字が増えれば必然的に HC 表が大きくなるからである．

4.1 PEM への応用

われわれはすでに，パターン抽出によるデータ圧縮法 (PEM)[3)〜5)]を開発している．PEM は，**図7**のように与えられたデータから圧縮に都合の良い文字列を選び出して，それに圧縮コード（図7の〈129〉や〈130〉）

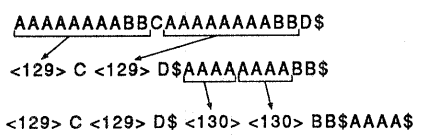

図 7 PEM による圧縮の過程
Fig. 7 Process of compression by PEM.

を割り振っていく．圧縮コードとして 129〜2,000 程度を用いるので，上述の「多くの値をもつ文字」の性質があり，CHT に適している．

また，PEM はその性質上7ビットデータのみに適用可能であった．今回，圧縮コードとして 257〜2,047 を用いることにしたので8ビットデータに対しても適用可能となった．これを，EPEM (Enhanced PEM) と呼ぶ．

表2の③が PEM による圧縮率である．圧縮率とは，「圧縮データのバイト数/元データのバイト数」である．

④は，PEM 適用後のデータ（各文字は 11 ビット）を，HC で記憶したものである．

⑤は，PEM の「選び出す文字列」の長さの計算において，各文字の大きさを1バイトでなく，HC のパターンの長さ（1〜19 ビット）で計算するように PEM を変更したものである．文字の大きさは，圧縮コードの置き換えを行うたびに再計算している．PEM 適用後のデータは HC で記憶してある．

これらの HC 表は CHT で記憶してある．その大きさ（表1の②に相当）を出現文字（圧縮コード）数で割った値は，平均 2.7 bit/コードであった．これは，従来方式の約 16% の大きさである．この値は，CHT の有効性を示している．

これらの圧縮率は，HC による圧縮率①や compress による圧縮率②に比べてかなり良いことがわかる．compress は UNIX のデータ圧縮用コマンドであり，Lempel-Ziv アルゴリズム[6)]により圧縮を行っている．

表 2 各種の圧縮法での圧縮率
Table 2 Compression ratio for various compression methods.

		prg 1	prg 2	prg 3	roff	tex	man	68 k	r 3 k
大 き さ	(byte)	79088	12121	2091	105596	92648	75461	77824	102400
Huffman①	(%)	63.0	61.6	66.4	60.9	62.2	73.7	67.0	64.3
Compress②	(%)	40.6	45.0	57.8	41.2	43.2	45.6	53.2	57.2
PEM③	(%)	34.6	37.7	50.0	37.2	36.4	—	—	—
EPEM④	(%)	29.4	32.7	44.8	31.8	31.1	33.1	41.9	47.4
EPEM⑤	(%)	27.7	32.1	43.9	30.1	29.3	32.1	40.7	45.5

4.2 compress への応用

compress についてもその処理要素 (9〜16 ビット) を，HC で記憶する実験を行ったが，圧縮率はかえって悪くなってしまった．これは，値が一度しか現れない処理要素が比較的多いためである．

5. ま と め

本論文で述べた CHT はわれわれが独自に開発したものであるが，パターンの代わりに長さを記憶するという考え方は文献 7) でもふれられている．しかし，CHT には文献 7) にある文字コードの配置の制約はない．また，処理できる文字コードの大きさに対する正確な検討を行っている．さらに，文字コードが極端に大きい場合についても有効であることを実例で示している．

HC はその性質上，適用分野の広いものである．しかし，「HC 表の大きさ」のために適用できなかった分野が多くあると思われる．本論文で述べた CHT は，HC の適用分野を広げるものと確信する．

また，応用で述べた EPEM は画期的な圧縮率を達成しており，実用データへの適用が増えることが期待される．

参 考 文 献

1) Dishon, Y.: Data Compaction in Computer System, *Computer Design*, April, pp. 85-90 (1977).
2) 宮川 洋，原島 博，今井秀樹：情報と符号の理論，p. 266，岩波書店，東京 (1982).
3) 河村知行：自動的なパターン抽出によるデータ圧縮法の提案，情報処理学会論文誌，Vol. 25, No. 6, pp. 1089-1094 (1984).
4) Kawamura, T.: Data Compression by Hardware PEM Using Multi Processor Elements, *Journal of Information Processing*, Vol. 9, No. 4, pp. 213-219 (1986).
5) 河村知行：パターン抽出によるビットデータ圧縮法，情報処理学会論文誌，Vol. 28, No. 9, pp. 995-997 (1987).
6) Ziv, J. and Lempel, A.: Compression of Individual Sequences via Variable-Rate Coding, *IEEE Trans.*, Vol. IT-24, No. 5, pp. 530-536 (1978).
7) 奥村晴彦：圧縮アルゴリズム入門，*C-MAGAZINE*, Vol. 3, No. 1, pp. 44-58 (1991).

(平成5年 4 月 11 日受付)
(平成5年10月 28 日採録)

河村 知行（正会員）

1953年生．1976年東京教育大学理学部応用数理学科卒業．1979年筑波大学大学院博士課程数学研究科中途退学．1979年徳山工業高等専門学校情報電子工学科助手．1981年同助教授，現在に至る．理学博士．計算機システム，計算機アーキテクチャ，アルゴリズムに興味をもつ

江口 賢和（正会員）

1946年生．1969年山口大学工学部電気工学科卒業．1971年同大学院電気工学専攻修士課程修了．1974年九州大学大学院電子工学専攻博士課程単位取得退学．工学修士．同年九州工業大学情報工学科助手．現在，徳山工業高等専門学校情報電子工学科助教授．データベースシステムにおける並列実行制御に関する研究に従事．電子情報通信学会会員．

重村 哲至（正会員）

1964年生．1987年豊橋技術科学大学工学部情報工学科卒業．1989年豊橋技術科学大学大学院工学研究科情報工学専攻修了．工学修士．1989年より徳山工業高等専門学校助手