PUC

Série : Monografias em Ciência da Computação
Nº 21/83

UM ESTUDO EM COMPRESSÃO DE DADOS

por

Antonio Carlos Pereira Maia

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 – CEP-22453
RIO DE JANEIRO – BRASIL

PUC/RJ - DEPARTAMENTO DE INFORMÁTICA

Série : Monografias em Ciência da Computação, Nº 21/83

Editor : Antonio L. Furtado Novembro , 1983

UM ESTUDO EM COMPRESSÃO DE DADOS*

por

Antonio Carlos Pereira Maia**

* Trabalho parcialmente financiado pela FINEP

** Aluno de Mestrado do Departamento de Informática

S U M Ā R I O

UM ESTUDO EM COMPRESSÃO DE DADOS

Antonio Carlos Pereira Maia

RESUMO

Técnicas de compressão de dados são ferramentas que podem e devem ser usadas em sistemas de informação para melhorar seu desempenho. Isto é possível com a redução do espaço de armazenamento necessário para a representação dos dados.

Este trabalho busca analisar as técnicas em compressão de dados descritas na literatura e comentar os benefícios e restrições associadas a seu uso.

Palavras-chave

Manutenção de Arquivos
Recuperação de Informações
Compressão de Textos
Compressão de Arquivos de Dados
Técnicas de Codificação
Armazenamento de Dados
Gerência de Dados

ABSTRACT

Data compression techniques are tools that can and must be used on information systems in order to improve their performance. This is possible by reducing the necessary storage space for data representation.

The paper tries to analyse the techniques described in the literature on data compression and discuss the benefits and constraints associated with their use.

Key-words

File Maintenance

Information Retrieval

Text Compression

Data File Compression

Coding Techniques

Data Storage

Data Management

1 - INTRODUÇÃO

Embora possa haver disponibilidade de memória secundária em uma instalação, um fato bastante comum é a necessidade crescente de mais área para atender à demanda provocada pelo desenvolvimento de aplicações em processamento de dados. Grande parte dessas aplicações compreende o armazenamento e transmissão de arquivos de dados, programas e textos.

Há uma série de técnicas que reduzem os requisitos de memória secundária para a representação dos dados e, conseqüentemente, os custos de armazenamento e de transmissão.

As técnicas de compressão existentes possibilitam reduções de até 98% ([1]) em aplicações em telemetria, nas quais há grande redundância, devido à forte correlação existente entre valores de dados sucessivos. Outras formas de redundância sucedem quando um ou mais valores ocorrem com altas freqüências ou variam em um domínio menor do que pode ser representado com seus formatos de armazenamento.

2 - ALGUNS CONCEITOS E TERMINOLOGIA

Alguns conceitos específicos são importantes no estudo de compressão de dados e merecem uma definição precisa.

Severance ([2]) distingue os seguintes termos relacionados:

- codificação de dados - processo pelo qual se mapeia uma coleção de unidades de codificação (i.e., um ou mais símbolos em uma representação de dados) em uma coleção de valores-códigos (i.e., um ou mais símbolos em uma segunda representação de dados), sendo um código o relacionamento existente entre as unidades de codificação e seus valores-códigos correspondentes. Se um mapeamento do código é um-para-um, existe um mapeamento inverso cujo processo é chamado decodificação;

- compactação de dados - forma de codificação de dados que reduz o tamanho do dado, preservando toda informação considerada relevante, através de um mapeamento não necessariamente um-para-um (p.ex., abreviação);

- compressão de dados - processo reversível de compactação de dados.

Outro conceito importante é o de razão de compressão de um código, que é a quantidade relativa de memória economizada pela codificação. Assim, se uma base de dados possui um comprimento L e a mesma base compactada possui um comprimento $\hat{L}$, a razão de compressão é $(L-\hat{L})/L$.

O comprimento de uma base de dados é definido por $L = m \sum_{i=1}^{n} \ell i . pi$, onde:

m = número total de ocorrências de unidades de codificação na base de dados

n = número de unidades de codificação distintas

ℓi e pi = comprimento e probabilidade de ocorrência correspondente a cada unidade de codificação.

E o comprimento comprimido da base de dados definido por $\hat{L} = n \sum_{i=1}^{n} \hat{\ell} i . pi$, onde:

$\hat{\ell}$ = comprimento do valor-código correspondente à unidade de codificação.

Podemos também definir a entropia de uma base de dados, que corresponde ao limite mínimo do comprimento do valor de código médio e é definido por $E = \sum_{i=1}^{n} Pi . \log_2 (1/pi)$, onde:

E = entropia da base de dados

3 - VANTAGENS E DESVANTAGENS DO USO DAS TÉCNICAS DE COMPRESSÃO

A compressão de dados reduz os custos de armazenamento e de transmissão e aumenta as velocidades de acesso e transferência de dados, bem como a velocidade de transmissão efetiva dos dados. Além disso, as operações de recuperação, *back-up*, ordenamento e intercalação podem ser mais eficientes, pois manipulam arquivos menores. Codificações especiais (ex: Encryptologia) podem ser aplicadas, visando à segurança de dados confidenciais.

Apesar dessas vantagens, tais técnicas não são amplamente utilizadas nos ambientes de desenvolvimento de sistemas de informação. Segundo Severance ([2]), três fatos ajudam a explicar este comportamento:

([1]) Os projetistas em geral subestimam o potencial de compressão possível para uma dada base de dados, e não conseguem avaliar bem suas implicações e benefícios;

([2]) A compressão de dados adiciona uma camada de complexidade ao projeto, implementação e operação de um sistema de informação, e os projetistas relutam em aceitar complexidades adicionais sem estarem seguros de benefícios substanciais;

([3]) A maior parte da literatura publicada aborda técnicas de compressão de dados voltadas para problemas específicos, normalmente encobertas por uma abordagem matemática.

Uma das maiores desvantagens é o acréscimo de processamento, com repercussão no tempo de resposta necessário às operações de compressão e descompressão. Uma camada de complexidade significativa é acrescentada quando é preciso desenvolver procedimentos para a implementação de técnicas que requerem manipulação de *bits*, por exemplo, o que normalmente é ineficaz e de difícil execução com linguagens de programação de alto nível. Mais tempo deve ser alocado nas diversas fases de desenvolvimento dos novos sistemas.

Manutenção nas tabelas de compressão e descompressão é necessária sempre que novos dados são armazenados nas bases de dados.

4 - ANÁLISE DOS MÉTODOS DE COMPRESSÃO

Segundo a literatura ([3],[4]), é comum dividir os métodos de compressão de dados em duas categorias: a primeira engloba os métodos dependentes da estrutura dos registros ou do conteúdo dos dados, e que deveriam ser empregados em aplicações específicas; e a segunda corresponde aos métodos que consideram uma base de dados como uma seqüência de *bits* e reduzem seu tamanho, ignorando sua estrutura e conteúdo. Esta última seria mais adequada a aplicações gerais e assim poderia ser implementada por *software*, microcódigo ou *hardware* de propósitos gerais. Porém, após uma verificação mais cuidadosa, nota-se que esta interface não é tão bem definida. Um exemplo ilustrativo é o método de supressão de caracteres repetidos, que tanto pode ser usado na supressão de zeros em um campo numérico de um registro formatado, como na supressão de brancos em um texto qualquer.

Por razões de eficiência, os arquivos comerciais são normalmente compostos de registros formatados. Estes são formados por campos de dados, cujos valores pertencem a domínios conhecidos e controlados. São, portanto, típicos para os métodos da primeira categoria. Porém, os campos de dados, por sua vez, nada mais são do que seqüências de *bits*, sobre os quais se poderia aplicar um ou mais métodos de propósitos gerais.

Podemos afirmar, portanto, que existem métodos típicos e exclusivos da primeira categoria. Os métodos da segunda categoria, porém, embora mais voltados para aplicações gerais, podem também ser combinados com os da categoria anterior na otimização da compressão de dados em aplicações específicas.

A seguir serão analisados individualmente os diversos métodos e variantes de interesse segundo uma seqüência lógica, em função de sua generalidade e/ou complexidade.

5 - MÉTODOS DEPENDENTES DA ESTRUTURA DOS ARQUIVOS

5.1 - ELIMINAÇÃO DE ITENS DE DADOS REDUNDANTES

Este método consiste na redução do tamanho de arquivos de dados, através da eliminação da redundância devida ao armazenamento de um mesmo item de dado em arquivos separados.

Para exemplificar uma técnica empregada com este objetivo, consideramos o procedimento de normalização para a quarta forma normal, no modelo relacional.

Seja uma relação não normalizada que contenha informação sobre cursos, professores e livros-texto. Seja um dado registro desta relação:

CURSO	PROFESSOR	LIVRO-TEXTO
FÍSICA	Prof. A Prof. B Prof. C	Livro-1 Livro-2 Livro-3

R

O significado deste registro é que o curso de FÍSICA pode ser ministrado por qualquer dos professores relacionados e usa todos os livros indicados. Suponhamos que os mesmos livros-texto serão usados pelo curso, independentemente do professor que o esteja oferecendo, ou seja, professores e livros-texto são independentes uns dos outros.

Convertendo esta estrutura em uma forma equivalente, obtemos a relação normalizada R_1:

R_1

CURSO	PROFESSOR	LIVRO-TEXTO
FÍSICA	Prof. A	Livro-1
FÍSICA	Prof. A	Livro-2
FÍSICA	Prof. A	Livro-3
FÍSICA	Prof. B	Livro-1
FÍSICA	Prof. B	Livro-2
FÍSICA	Prof. B	Livro-3
FÍSICA	Prof. C	Livro-1
FÍSICA	Prof. C	Livro-2
FÍSICA	Prof. C	Livro-3

É óbvio que esta relação contém bastante redundância, o que causa problemas nas operações de atualização, além de desperdiçar espaço de armazenamento.

Como, na relação R_1, professor é multidependente de curso, e o mesmo acontece em relação a livro-texto, i.e., um curso determina obrigatoriamente um conjunto de professores e um conjunto de livros-texto, ela pode ser decomposta em duas relações, R_1' e R_1'':

R_1'

CURSO	PROFESSOR
FÍSICA	Prof. A
FÍSICA	Prof. B
FÍSICA	Prof. C

R_1''

CURSO	LIVRO-TEXTO
FÍSICA	Livro-1
FÍSICA	Livro-2
FÍSICA	Livro-3

Com estas projeções, as duas relações resultantes encontram-se em 4FN, e o problema de redundância está sanado.

5.2 - CONVERSÃO DE ROTAÇÃO HUMANA PARA ROTAÇÃO COMPACTADA

Os campos de dados normalmente são armazenados na forma escrita das linguagens naturais. Esta forma de armazenamento não é

normalmente a mais adequada, pois pode conter mais caracteres do que o necessário.

Os campos de datas são freqüentemente armazenados na forma DDMMAA (p.e., 060555 representando 06 de maio de 1955), o que ocupa 6 *bytes* em um arquivo. Porém, o dia necessita de apenas 5 *bits* para ser representado, o mês e o ano requerem 4 e 7 *bits*, respectivamente. Usando-se esta forma alternativa, são necessários apenas 16 *bits* para armazenar qualquer data.

Outra forma alternativa é representar datas segundo o calendário Juliano. Selecionando-se uma data como um ponto base, datas comprimidas podem ser representadas como uma distância em dias a partir daquela origem. Em 1582, Joseph Scalizer propôs esta forma para uso astronômico. A data escolhida como ponto base foi 1 de janeiro de 4713 BC. Para as aplicações comerciais pode ser escolhida uma data mais recente para ponto base. Assim, um código de 16 *bits* poderia ser usado para representar datas que variam em um período de 175 anos.

5.3 - ELIMINAÇÃO DE ESPAÇO VAZIO

Um fato comum em arquivos comerciais é a reserva de espaço para itens de dados repetitivos, p.e., um campo reservado para cada dependente. Obviamente, este procedimento induz a uma grande quantidade de espaço vazio nos arquivos. A solução normalmente adotada é a organização dos arquivos em registros de comprimento variável.

Um esquema para gerenciar os itens de dados presentes em um dado registro de comprimento variável poderia utilizar um mapa de *bits* conforme a figura 1.a (sugerida em 4).

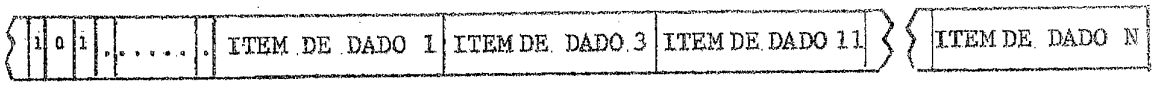
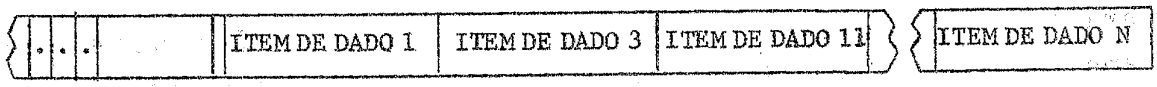


Figura 1
(a) ↳ mapa de *bits* indicando que itens de dados estão presentes

^{0} = Item de dado ausente
^{1} = Item de dado presente



(b) ↳ mapa de endereços

Um esquema adequado para o caso de os próprios itens de dados terem comprimentos variáveis é apresentado na figura 1.b. Nesta hipótese, o mapa de *bits* seria substituído por um mapa de endereços. Este seria um vetor de endereços onde cada campo indicava a posição no registro onde começa o item de dado correspondente. A localização de um item de dado seria calculada a partir dos comprimentos de cada item de dado presente e que o antecede no registro.

5.4 - CODIFICAÇÃO DE ITENS FREQÜENTEMENTE USADOS

Este método é usado para codificar um conjunto pequeno de unidades de codificação que aparecem com bastante freqüência em um texto ou em uma base de dados.

Exemplos comuns são campos como SEXO, NÍVEL DE ESCOLARIDADE, ESTADO CIVIL, que correspondem a um conjunto pequeno de valores, e, assim, podem ser codificados com um número muito pequeno de *bits*.

Martin ([4]) sugere que nomes de pessoas podem ser codificados em um *byte* de oito *bits*. Como existe uma variedade de nomes imprevisíveis e obviamente não comuns, poderíamos codificar 255 nomes mais freqüentes e reservar um *byte* de exceção para indicar que os

bytes seguintes representam um nome por extenso, não pertencente ao conjunto dos 255 mais freqüentes.

Como é possível que a tabela de substituição (código-nome) com 255 entradas possa ser considerada muito grande, uma vez que deve ser mantida em memória durante o processo de codificação/decodificação, o número de *bits*, e conseqüentemente de entradas, da tabela de substituição poderia ser reduzido, e os *bits* restantes poderiam ser usados para, p.ex., indicar o sexo ou o estado civil.

5.5 - SUBSTITUIÇÃO DE TEXTO

A idéia básica neste método é a de que palavras ou frases comuns em um dado tipo de texto podem ser substituídas por caracteres-código. Supondo que nosso arquivo armazena *bytes* de 8 *bits*, temos 256 combinações possíveis de *bits* que poderiam ser usadas na seguinte estratégia para a codificação das palavras ou frases:

1. As 200 palavras ou chaves mais comuns seriam representadas pelos números de 0 a 199;
2. Os números de 200 a 231 indicariam que este caractere sozinho não dá a palavra em questão, mas que o caractere seguinte é também necessário;
3. Os números de 232 a 256 indicariam que a palavra ou frase está por extenso em código EBCDIC.

Uma outra idéia consiste em fazer resumos de textos, que concentrariam o significado essencial dos mesmos. Estes resumos poderiam ser produzidos por especialistas ou automaticamente, isolando-se as sentenças mais significativas de um dado texto de acordo com as freqüências de ocorrência das palavras que as compõem e imprimindo-as na ordem.

5.6 - SUPRESSÃO DE NULOS

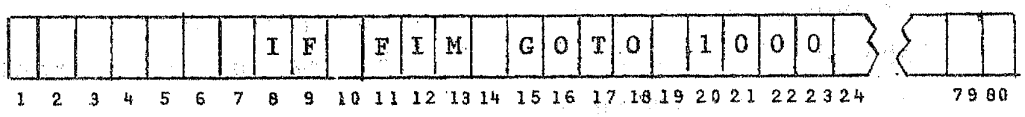
Em arquivos de dados comerciais, em fontes de programas e

em arquivos de textos de um modo geral é bastante comum a presença de grandes quantidades de brancos e zeros que formam seqüências contínuas. Com custos de processamento modestos é possível comprimir estas seqüências e obter consideráveis reduções de espaços de armazenamento.

Uma das técnicas de implantação da supressão de nulos mais comumente empregadas consiste em inserir no texto um caractere que indica a presença de uma dada seqüência de nulos a ser suprimida e de um contador que indica o comprimento desta seqüência.

Os caracteres que forem usados como indicadores do tipo de nulo devem ser caracteres não válidos no texto, o que facilita a decodificação posterior. Porém, se tal caractere não existir, qualquer um que seja infreqüente pode ser escolhido, e sua ocorrência duplicada sempre que este aparecer como dado. No exemplo da figura 2, foram utilizados os caracteres # e %.

Texto original



Texto comprimido

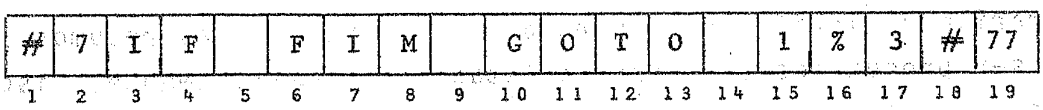


Figura 2. Supressão de nulos - #... 5, % ...0

Se um *byte* de oito *bits*, p. ex., for usado como contador do comprimento de seqüência, é possível representar a supressão de até 255 nulos por um único par < indicador, contador >. Se o número de nulos for maior que 255, são usados tantos pares quantos forem necessários.

Em transmissão de mensagens é comum a utilização de uma técnica de supressão dos brancos do início e fim do registro. Um par de contadores é inserido no início de cada registro: o primeiro indica o número de brancos que precedem o primeiro caractere não branco; e o segundo indica o número de caracteres significativos do registro. O texto comprimido resultante do emprego desta técnica ao texto da figura 2 seria:

7	16	I	F		F	I	M		G	O	T	O		1	0	0	0
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Martin (4) sugere um esquema de compressão que usa um grupo de caracteres para indicar que o caractere seguinte é repetido. Sua técnica baseia-se no fato de que a codificação de caracteres EBCDIC, de oito *bits*, possui uma quantidade de combinações de *bits* bem superior à necessária para o conjunto de caracteres efetivamente usado nos arquivos. Considere um conjunto de dados que não utilize letras minúsculas. Neste caso, o segundo *bit* do código efetivamente utilizado é sempre "1", de modo que todos os códigos com o *bit* "0" nesta posição podem ser usados para designar a supressão de nulos. Os 7 *bits* restantes, como apresentado na figura 3, podem ser usados para representar tanto o tipo do nulo quanto o comprimento da seqüência. No esquema proposto na figura 3, é possível representar a supressão de até 64 nulos de um entre dois tipos.

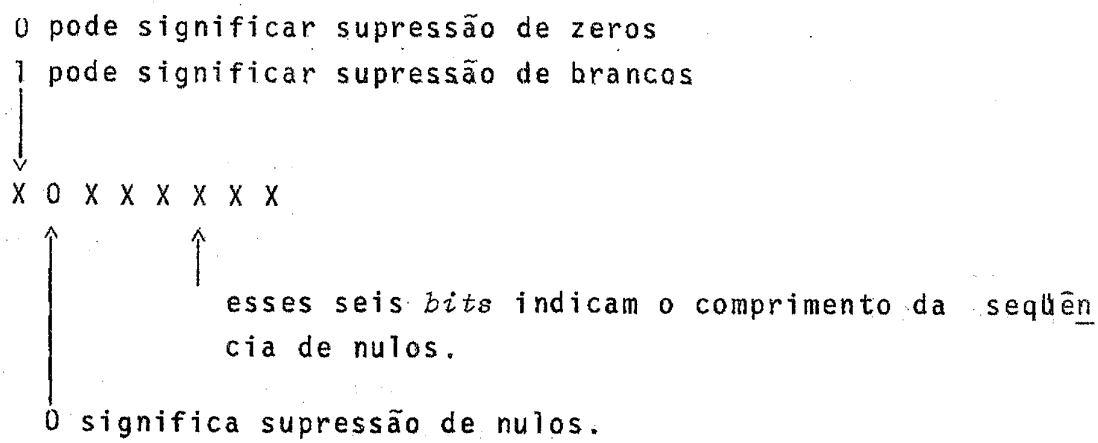


Figura 3. Supressão de nulos com EBCDIC.

5.7 - CODIFICAÇÃO DIFERENCIAL

Conforme Severance ([2]), a codificação diferencial envolve a reposição de uma unidade de codificação por um valor-código que define um relacionamento ou com uma unidade de codificação anterior ou com um valor padrão escolhido.

Uma técnica de codificação diferencial, usada principalmente em aplicações do tipo controle de processo - onde um computador dedicado colhe e registra periodicamente leituras referentes às medidas de fenômenos que estejam sendo controlados, de tamanho uniforme e que não tendem a sofrer variações bruscas - consiste em representar essas leituras por suas diferenças em relação à leitura anterior. Com esta compressão há uma grande redução na quantidade de dados gravados ou transmitidos a uma estação remota.

Em aplicações comerciais, técnicas de codificação diferencial são usadas quando essas aplicações lidam com valores ordenados e lidos seqüencialmente.

Martin ([4]) apresenta um exemplo para uma lista telefônica. Neste exemplo, apresentado na figura 4, os elementos dos itens de dados, em uma fase inicial, são abreviados substituindo os prefixos comuns por um dígito, técnica chamada compressão de prefixos.

Em uma segunda fase, os primeiros nomes comuns são substituídos por um código, estabelecido por uma tabela de codificação.

FIGURA 4 - COMPRESSÃO DE VALORES ORDENADOS

ORIGINAL	COMPRESSÃO FASE 1	COMPRESSÃO FASE 2	
WESTCOTT D.L.	WESTCOTT D L	WESTCOTT D L	Tabela de
WEST JOHN C.	3T JOHN C	3T 40C	Substituição
WEST JOHN J.JR.	9 J JR	9J ‡	12=ETHEL
WEST PHILLIP	5PHILLIP	561	40=JOHN
WESTBROOK C.	4BROOK C	4BROOK C	61=PHILLIP
WESTCOTT C. RAYMOND	4COTT C RAYMOND	4COTT C 65	65=RAYMOND
WESTERMAN C.R. MRS.	4ERMAN C R MRS	4ERMAN C R *	69=ROBERT
WESTERN THOMAS C.	6N THOMAS C	6N80C	80=THOMAS
WESTERN T.P.MRS.	8T P MRS	8T P*	89=WILLIAM
WESTERN UNION INC.	8UNION INC	8UNION@	
WINGATE WILLIAM	WINGATE WILLIAM	WINGATE 89	
WINSTON ROBERT	3STON ROBERT	3STON69	
WINTRAIL ETHEL E.MRS.	3TRAL ETHEL E MRS	3TRAL12E*	

As palavras descritivas comuns foram substituídas por caracteres especiais:

* = MRS.
‡ = JR.
@ = INC.

Date (9) sugere uma outra aplicação com valores-chave ordenados. A técnica de compressão de prefixos, desta vez, é implementada substituindo-se o prefixo de uma chave comum ao prefixo da chave comum ao prefixo da chave anterior por um contador do número de caracteres coincidentes, conforme o seguinte exemplo:

JOHNSON		(0) JOHNSON
JONAH	⟹	(2) NAH
JONES		(3) ES
JORGENSON		(4) REENSON

O procedimento de decodificação, da mesma forma que o de codificação, exige leitura seqüencial, de modo que o valor da chave precedente esteja novamente disponível.

Neste caso, uma compressão ainda maior poderia ser obtida se os caracteres finais de cada chave que não fossem necessários para diferenciá-la das adjacentes fossem simplesmente truncados.

6 - MÉTODOS DE CODIFICAÇÃO DE CADEIAS DE CARACTERES

6.1 - INTRODUÇÃO

Esses métodos enfocam a compactação de cadeias de caracteres de comprimentos arbitrários. Os mais simples dividem a seqüência de caracteres em subcadeias unitárias. À medida que manipulam cadeias de comprimentos maiores, os métodos se tornam mais eficientes, ou seja, alcançar maiores taxas de compressão, à custa de maiores volumes de processamento e de memória principal para tabelas e algorítmos.

6.2 - COMPRESSÃO DE CARACTERES

Um esquema de compressão de caracteres explora o fato de os códigos de armazenamento, como o EBCDIC, apresentarem uma quantidade possível de formatos de armazenamento muito superior à realmente necessária para o conjunto de caracteres efetivamente usados nos arquivos.

Uma análise da figura 5 mostra que são necessários apenas os seis últimos *bits* para diferenciar os caracteres alfabéticos maiúsculos e os numéricos entre si. A remoção dos dois primeiros *bits*

reduziria a área de armazenamento em 25%. Uma redução de 37% seria obtida se os dados fossem estritamente alfabéticos, com a utilização de 5 *bits* para representar cada caractere.

POSIÇÕES DE *BITS* 0 E 1

Posições de *bits* 4,5,6 e 7	POSIÇÕES DE *BITS* 2 e 3				POSIÇÕES DE *BITS* 2 e 3				POSIÇÕES DE *BITS* 2 e 3				POSIÇÕES DE *BITS* 2 e 3			
	00	01	10	11	00	01	10	11	00	01	10	11	00	01	10	11
0000					ƀ	&	-					~	{	}	#	0
0001							/		a	j	s		A	J		1
0010									b	k	t		B	K	S	2
0011									c	l	u		C	L	T	3
0100									d	m	v		D	M	U	4
0101									e	n	w		E	N	V	5
0110									f	o	x		F	O	W	6
0111									g	p	y		G	P	X	7
1000									h	q	z		H	Q	Y	8
1001									i	r			I	R	Z	9
1010					ç	!		:								
1011					.	$	,	#								
1100					<	*	%	@								
1101					(	)	–	'								
1110					+	;	>	=								
1111					|	¬	?	"								

FIG. 5 EBCDIC

Uma compressão maior pode ser obtida com um código que empregue um número variável de *bits* por caractere. Assim, os caracteres com probabilidades de ocorrências mais altas corresponderiam a valores de código mais curtos e vice-versa.

6.3 - COMPRESSÃO DE CADEIAS DE CARACTERES

6.3.1 - Através de código de comprimento fixo

Dado um universo de N unidades de codificação distintas, é sempre possível atribuir números binários de 0 a N-1 a essas unidades de codificação e assim construir um código de comprimento fixo de comprimento $\bar{\ell}$, sendo $\bar{\ell}$ igual ao menor inteiro maior ou igual a $\ell og_2 N$.

A codificação e decodificação com os códigos de comprimento fixo é simples e direta. A codificação é feita através de pesquisa em memória principal, e a decodificação através do uso do valor-código para calcular a posição relativa da unidade de codificação na tabela de código.

6.3.2 - Através do código de Huffman

Este código visa aumentar a compressão de dados através do emprego de um número variável de *bits* por unidade de codificação. Ele é baseado no fato de que as unidades de codificação, sejam elas caracteres ou cadeias de caracteres, ocorrem com freqüências diferentes. Assim, procura atribuir valores-código de comprimentos menores às unidades de codificação mais freqüentes e valores-código de comprimentos maiores às menos freqüentes.

Para uma dada relação de unidades de codificação com suas probabilidades correspondentes, é possível gerar vários códigos de comprimento variável usando-se o conceito exposto. Porém, os códigos de Huffman levam à redundância mínima, ou seja, ao menor comprimento médio possível.

O código de Huffman pode ser obtido através da construção de uma árvore binária. Parte-se de uma lista das unidades de codificação ordenadas por suas probabilidades de ocorrência. As duas unidades de probabilidades menores são retiradas da lista e substituídas no conjunto, por sua soma. Atribui-se ao ramo da unidade de

maior valor de probabilidade de ocorrência o valor ϕ, e ao de menor valor de probabilidade de ocorrência o valor 1. Esta operação é repetida até que uma única unidade permaneça como a raiz da árvore recém-construída. A figura 6 apresenta a árvore resultante, e a tabela I o código resultante para o exemplo das dez palavras mais comuns em inglês ([2]).

TABELA I

EXEMPLO DO CÓDIGO DE HUFFMAN

i	UNIDADE DE CODIFICAÇÃO	PROBABILIDADE	VALOR-CÓDIGO	COMPRIMENTO
1	THE-	.270	00	2
2	OF-	.170	010	3
3	AND-	.131	100	3
4	TO-	.099	101	3
5	A-	.088	111	3
6	IN-	.074	0110	4
7	THAT-	.052	1100	4
8	IS-	.043	1101	4
9	IT-	.040	01110	5
10	ON-	.033	01111	5

Comprimento médio: 2x.270+3x.170+3x.131+3x.099+3x.088+4x.074+4x.052 + 4x.043+5x.040+5x.033 = 3.045 *bits*.

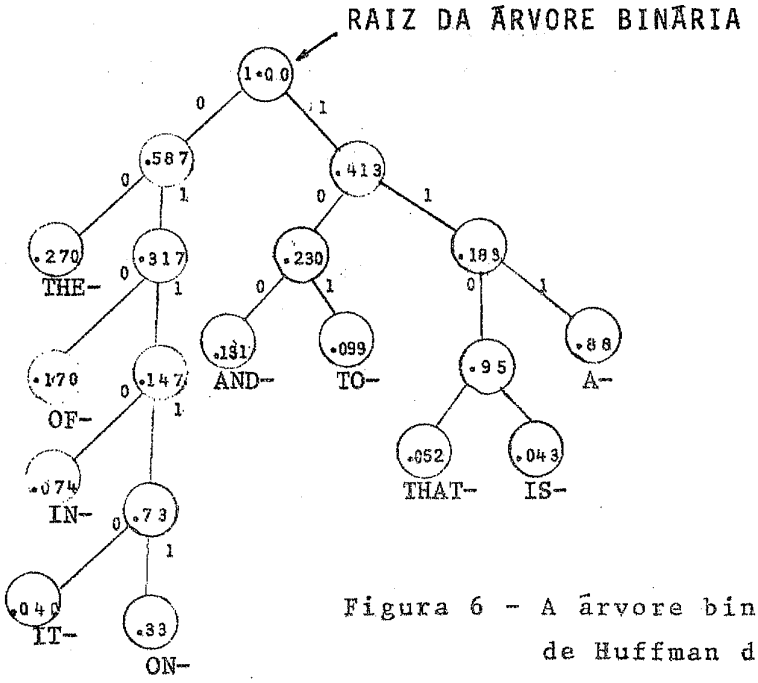


Figura 6 - A árvore binária do código de Huffman do exemplo.

A tabela II apresenta um código alternativo formado pela complementação do código da tabela I. Como o comprimento médio deste código alternativo é idêntico ao de Huffman, pode-se concluir que há mais de um código ótimo.

TABELA II

CÓDIGO ALTERNATIVO ÓTIMO PARA O EXEMPLO PROPOSTO

i	UNIDADE DE CODIFICAÇÃO	PROBABILIDADE	VALOR-CÓDIGO	COMPRIMENTO
1	THE-	.270	11	2
2	OF-	.170	101	3
3	AND-	.131	011	3
4	TO-	.099	010	3
5	A-	.088	000	3
6	IN-	.074	1001	4
7	THAT-	.052	0011	4
8	IS-	.043	0010	4
9	IT-	.040	10001	5
10	ON-	.033	10000	5

A tabela III contém um outro código de comprimento variável para o exemplo proposto. Este código apresenta um comprimento médio maior, ou seja, é menos eficiente, porém é muito mais simples de se codificar. Este código segue as seguintes regras para a determinação do número de *bits* dos valores-código:

1. Se o primeiro *bit* é 0, o valor-código tem 1 *bit*;
2. Se os dois primeiro *bits* são 1 0, o valor-código tem 3 *bits*;
3. Caso contrário, o valor-código tem 5 *bits*.

TABELA III

UM CÓDIGO ALTERNATIVO NÃO ÓTIMO, MAS DE FÁCIL CODIFICAÇÃO PARA O EXEMPLO PROPOSTO

i	UNIDADE DE CODIFICAÇÃO	PROBABILIDADE	VALOR-CÓDIGO	COMPRIMENTO
1	THE-	.270	0	1
2	OF-	.170	100	3
3	AND-	.131	101	3
4	TO-	.099	11000	5
5	A-	.088	11001	5
6	IN-	.074	11010	5
7	THAT-	.052	11011	5
8	IS-	.043	11100	5
9	IT-	.040	11101	5
10	ON-	.033	11110	5

Comprimento médio: 1x.270+3x.170+3 x. 131+5x.099+5x.088 + 5x.074+5x.052+5 x. 043+5x.040+5x.033= 3.318 *bits*.

Observando-se a formação dos códigos de comprimento variável, nota-se que os mesmos alçançam reduções cada vez mais drásticas quanto mais assimétrica for a distribuição das probabilidades, com o comprimento médio do valor-código tendendo para a entropia. No caso contrário, quanto mais homogênea for esta distribuição, menor será a redução, e, assim, se utilizamos o código de Huffman para um conjunto de unidades de codificação equiprováveis, obteremos um código de comprimento fixo, sem, portanto, alcançar qualquer redução adicional.

6.3.3 - Através de N. gramas

Snyderman e Hunt ([5]) apresentam um trabalho de compressão de unidades de codificação formadas de até dois caracteres (chamadas digramas). Unidades de codificação com N caracteres são ditas N.gramas. O conceito empregado explora o fato de que a maioria dos dados armazenados usa somente 88 das 256 representações de um código de 8 *bits*, deixando assim 168 valores de código hexadecimais disponíveis para uso em representações de pares de caracteres.

O processo de compactação exposto é realizado em três fases. Na primeira fase, cada caractere no texto é traduzido para um código hexadecimal compactado. Este código alternativo agrupa os 88 símbolos mais usados no intervalo 00 a 57 hexadecimal, deixando 168 representações contíguas no fim da escala, de 58 a FF hexadecimal.

A segunda fase consiste na combinação de caracteres em pares, de modo que um único valor de código hexadecimal represente ambos. A compactação é feita testando-se em seqüência cada caractere para determinar se ele é de um determinado tipo dito mestre (conforme figura 7). Em caso positivo e se o caractere seguinte for de um tipo dito combinadores, ambos são representados por um valor codificado formado pela adição do valor do código do componente combinado com o valor base correspondente ao componente mestre. Caso contrário, os valores dos códigos dos caracteres envolvidos são passados inalterados.

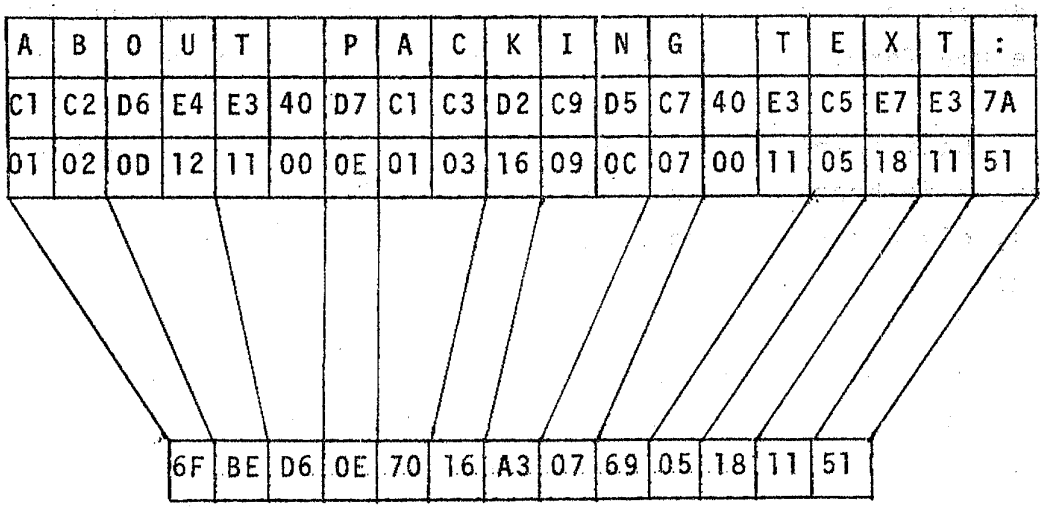
FIGURA 7 - CODIFICAÇÃO DIGRAMA

CARACTERES MESTRES		CARACTERES COMBINADORES		VALORES NÃO COMBINADORES						PARES COMBINADOS			
SÍMBOLO	VALOR BASE	SÍMBOLO	CÓDIGO HEX.	SÍMBOLO	CÓDIGO HEX	SÍMBOLO	CÓDIGO HEX	SÍMBOLO	CÓDIGO HEX	SÍMBOLO	CÓDIGO HEX	SÍMBOLO	CÓDIGO HEX
B	58	B	00	J	15	q	2B	<	41	BB	58	AB	6D
A	6D	A	01	K	16	r	2C	(	42	BA	59	AA	6E
E	82	B	02	Q	17	s	2D	+	43	BB	5A	AB	6F
I	97	C	03	X	18	t	2E	&	44	BC	5B	AC	70
O	AC	D	04	Y	19	u	2F	|	45	BD	5C	↓	↓
N	C1	E	05	Z	1A	v	30	$	46	BE	5D	AW	81
T	D6	F	06	a	1B	w	31	*	47	BF	5E	EB	82
U	EB	G	07	b	1C	x	32	)	48	BG	5F	EA	83
		H	08	c	1D	y	33	;	49	BH	60	↓	↓
		I	09	d	1E	z	34	-	4A	BI	61	EW	96
		L	0A	e	1F	¢	35	/	4B	BL	62	IB	97
		M	0B	f	20	¦	36	,	4C	BM	63	↓	↓
		N	0C	g	21	2	37	%	4D	BN	64	OB	AC
		O	0D	h	22	3	38	_	4E	BO	65	↓	↓
		P	0E	i	23	4	39	¦	4F	BP	66	NB	C1
		R	0F	j	24	5	3A	?	50	BR	67	↓	↓
		S	10	k	25	6	3B	:	51	BS	68	TB	D6
		T	11	l	26	7	3C	#	52	BT	69	↓	↓
		U	12	m	27	8	3D	@	53	BU	6A	UB	EB
		V	13	n	28	9	3E	'	54	BV	6B	↓	↓
		W	14	o	29	¢	3F	=	55	BW	6C	WW	FF
				p	2A	.	40	"	56				
								<	57				

A terceira fase consiste na substituição dos caracteres originais pela representação do caractere único resultante.

A figura 7 apresenta um exemplo de compressão de um texto ([5]).

FIGURA 8 - EXEMPLO DE COMPRESSÃO



O arranjo dos caracteres na tabela da figura 7 é função da freqüência dos caracteres na língua inglesa. Outros fatores também considerados são a presença de letras minúsculas, caracteres especiais e a posição relativa de letras dentro das palavras. Existem, porém, muitas outras possibilidades, conforme apresentado na figura 9.

FIGURA 9 - POSSÍVEIS ARRANJOS DE CARACTERES

POSSÍVEIS ARRANJOS DE CARACTERES MESTRES/COMBINADORES	POSSIBILIDADES
4 mestres com 42 combinadores	168
5 mestres com 33 combinadores	168
6 mestres com 28 combinadores	168
7 mestres com 24 combinadores	168
8 mestres com 21 combinadores	168
9 mestres com 18 combinadores	168
10 mestres com 16 combinadores	168
11 mestres com 15 combinadores	168
12 mestres com 14 combinadores	168

A compressão de dados com codificação de digramas é limitada a uma redução máxima de 50%, o que é alcançado somente quando todo caractere é emparelhado em um digrama. Com dados de produção os autores alcançaram, com o arranjo apresentado, uma taxa de compressão de 35%.

6.3.4 - Através de códigos numéricos

Hahn ([6]) apresenta uma técnica eficiente para a compressão de cadeias de caracteres em códigos numéricos, através da utilização de uma forma de codificação de comprimento variável.

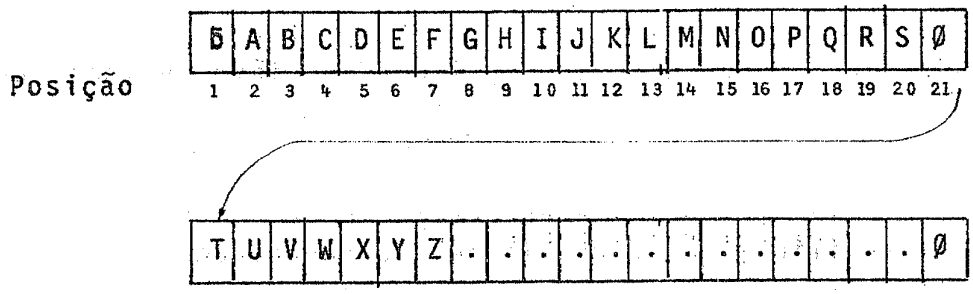
Basicamente, a compressão consiste em uma fase inicial de supressão de brancos e, em seguida, da codificação dos símbolos restantes, divididos em grupos de tamanho fixo (N), em número de ponto fixo.

Os símbolos que compõem os grupos são mantidos em tabelas de tamanho B-1, onde B é calculado de modo a produzir uma boa razão de compressão em função do tamanho da palavra da máquina. Os símbolos mais freqüentes são colocados na primeira tabela (dita dicionário primário), os próximos B-1, símbolos mais freqüentes, na segunda tabela, e assim sucessivamente. A B$^{\underline{\text{ésima}}}$ posição de cada tabela é reservada a um caractere de escape que permite a extensão do tamanho do dicionário.

Uma cadeia de símbolos com posições P_1, P_2,..., P_n é codificada como um único número de ponto fixo = $P_1 \times B^{n-1} + P_2 \times B^{n-2} + \ldots + P_n$. Um número qualquer de símbolos distintos poder ser codificado graças ao caractere de escape. Assim, qualquer caractere que não um dos B-1 primeiros do dicionário primário é representado pelo caractere de escape, codificado como zero, seguido pela posição do caractere na tabela seguinte, ou assim sucessivamente até que se localize sua posição.

Seja, por exemplo, codificar a seguinte cadeia de caracteres: ßßßßßßßEXEMPLOßOITO. As tabelas de tamanho B = 21 são apresentadas na figura 10. Os grupos são formados por 7 símbolos (N=7).

FIGURA 10 - EXEMPLO DO MÉTODO DE HAHN



Os símbolos seriam codificados como dois número de ponto fixo: o primeiro com valor igual a: $6x21^6 + 0x21^5 + 2x21^4 + 6x21^3 + 14x21^2 + 17x21 + 13 = 515.047.798$; e o segundo com o valor = $16x21^6 + 1x21^5 + 16x21^4 + 10x21^3 + 0x21^2 + 1x21 + 16 = 1.379.546.380$. Note que os símbolos T e U possuem as posições 22 e 23 e são codificados por 0,1 e 0,2 respectivamente. Um símbolo na posição 45, p.ex., seria codificado por 0,0,3.

O registro comprimido seria armazenado da seguinte forma:

7 12 515047798 1379546380

onde a primeira palavra indica o número de brancos (7) que precedem o primeiro caractere não branco e o número de símbolos (12) codificados.

Os valores de B e N são escolhidos de forma a minimizar o número de *bits*/caractere necessário para codificar o texto-fonte. Obviamente, o máximo de compressão vai ser obtido para os primeiros B-1 caracteres. O valor ótimo de B, para um dado valor de N, é o maior número de ponto fixo que pode ser armazenado em uma palavra.

Para textos em inglês, em uma máquina de 32-*bits*, Hahn encontrou a combinação ótima para B=21 e N=7, com uma entropia de 2.01 *bits*/caractere. Sem a supressão de brancos, o valor da entropia

subiria para 4.7 *bits*/caractere, valor este comparável aos 4.2 *bits*/caractere obtidos pelo código de Huffman para o mesmo tipo de texto.

6.3.5 - Através de códigos de comprimento variável e incremento fixo.

Este tipo de codificação, proposto por Heap ([10]), procura combinar a facilidade de gerência de códigos de comprimento fixo com a máxima compressão alcançada com códigos de comprimento variável. Esta técnica é particularmente indicada para palavras ou termos em dados textuais, onde os termos distintos são muito numerosos e se distribuem assimetricamente.

O esquema proposto sugere que, em um vocabulário de N termos distintos, os mais freqüentes recebam códigos de comprimento N_1 que indexarão uma tabela de tamanho 2^{N_1-1}. Os seguintes mais freqüentes receberão códigos de comprimento N_2, sendo $N_2>N$, e indexarão uma tabela de tamanho 2^{N_2-1} e assim sucessivamente.

Após experimentos com valores variados de N_1, N_2,..Nr, Heaps concluiu que códigos com comprimento (3, 6, 9, 12, ...) e (4,8,12, 16, ...) são adequados e fáceis de adaptar em máquinas com caracteres de 6 e 8 *bits*, respectivamente.

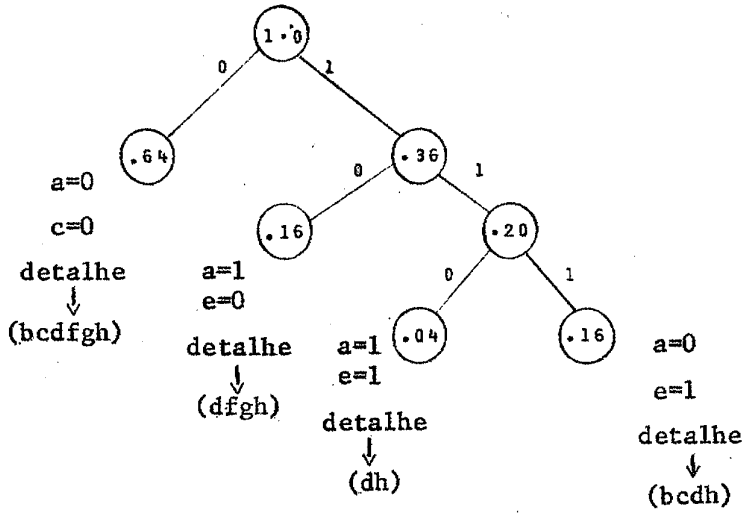
7 - MÉTODOS DE COMPRESSÃO DE DADOS DECIMAIS

Chen e Ho ([7]) reabrem a discussão em torno de aritmética decimal em computadores e abordam um dos aspectos negativos de seu uso: a ineficiência de armazenamento. O trabalho por eles apresentado propõe um esquema de compressão de números armazenados em rotação BCD. Este esquema envolve a compressão de 2 dígitos BCD em 7 *bits* e de 3 dígitos em 10 *bits*, através de um algoritmo baseado na combinação de duas variáveis de codificação de comprimento fixo.

Na rotação BCD, um dígito, A é representado por uma cadeia de 4 *bits* (a,b,c e d), tal que A = 8a+4 +2 +d, sendo que 80% dos estados (0-7) podem ser representados por apenas 3 *bits* (bcd). Dos estados restantes, dois *bits* são realmente necessários(ad) e os outros dois são predizíveis (b=0 e c=0), podendo ser omitidos. Assim, um dígito pode ser visto como composto de um *bit* indicador(a) de sua magnitude (se a=0, o dígito tem valor menor ou igual a 7) e de uma segunda parte, de detalhe: (b,c e d) para dígitos pequenos e (d) para os maiores (onde a=1).

Seja um par de dígitos A,B= (abcd) (efgh); seus *bits* indicadores (a,e) originam quatro estados indicadores de probabilidades de ocorrências distintas e que podem ser codificados em um código de Huffman, conforme a figura 11.

FIGURA 11 - REPRESENTAÇÃO EM ÁRVORE BINÁRIA DO CÓDIGO DE HUFFMAN GERADO



Combinando-se os códigos gerados para os estados indicadores com os campos de detalhes correspondentes (também indicados na figura 11), obtemos um código de comprimento fixo de 7 *bits*. Este código está representado na figura 13, sendo os *bits* resultantes chamados (pqrstuv).

FIGURA 12 - ESQUEMA DE COMPRESSÃO DE DOIS DÍGITOS BINÁRIOS (abcd) (efgh) em 7 *bits* (pqrstuv)

a	e	p	q	r	s	t	u	v	COMENTÁRIOS
0	0	0	b	c	d	f	g	h	
1	0	1	0	c	d	f	g	h	b=0
0	1	1	1	1	d	b	c	h	tu=bc
1	1	1	1	0	d	b	c	h	tu=bc,b=0
		Campo Indicador			Campo de Detalhe				

O mapeamento dos dois dígitos binários BCD em 7 *bits* e o mapeamento inverso podem ser representados pelos seguintes algoritmos:

Codificação:

pqrstuvw = abcdfgh se e=0 (80%)
= 11 ādbch em caso contrário

Decodificação:

abcdefgh = pqrsotuv se p.q = 0 (80%)
= r̄tus 1000 em caso contrário

O trabalho também apresenta o esquema para se codificar três dígitos em campos de dez *bits*.

Vale salientar que o esquema proposto pelos autores não requer aritmética, somante lógica, deleções, deslocamentos e inserções.

Smith ([8]), em uma análise do trabalho de Chen e Ho, propõe um esquema alternativo a esse método. Ele propõe que simplesmente se codifiquem campos de dois e três dígitos diretamente em binário. Este esquema teria a vantagem de manter a ordenação relativa de

grupos de números (por valor) e de permitir alguns cálculos aritméticos através de *hardware* binário, sem necessitar decodificação.

8 - CONCLUSÕES

As diversas técnicas de compactação apresentadas estabelecem um compromisso entre o grau de compressão alcançada em um arquivo e os custos de processamento associados à redução e expansão do mesmo.

Algumas variáveis afetam a escolha da melhor alternativa para uma base de dados específica, por exemplo, a quantidade e o tipo de redundância que ela contém, seu tamanho e a disponibilidade de área de armazenamento para tabelas de códigos e, inclusive, a eficiência e complexidade da técnica considerada.

A natureza e a freqüência de recuperações e atualizações devem também ser consideradas. Assim, técnicas como codificações numéricas e digramas são muito sensíveis às alterações nos dados. Quando a recuperação dos dados é preponderante, é interessante a utilização de códigos de comprimento fixo. Por possuírem campos e limites fixos, facilitam o acesso direto a registros, a compressão e descompressão seletiva de campos e a pesquisa de dados com chaves comprimidas sem necessidade de descompressão.

Técnicas como a supressão de nulos se caracterizam pela facilidade de implementação. Não necessitam de tabelas e alcançam taxas de compressão de até 70%.

Quando a base de dados possui unidades de codificação que se comportam segundo distribuições irregulares de suas ocorrências, os códigos de comprimento variável chegam a taxas de compressões significativamente superiores às dos códigos de comprimento fixo, especialmente o código de Huffman, que atinge a compressão ótima.

A técnica de compressão de Huffman requer a estimativa das probabilidades das ocorrências das unidades de codificação, o que se

torne ainda mais complexo quando essas probabilidades variam ao longo do tempo, como no caso de transmissão e recepção de mensagens em um equipamento de comunicações.

O código de incremento fixo e tamanho variável, devido à sua maior simplicidade e à ausência dos problemas de sincronização, é uma alternativa nas situações onde o código de Huffman seria recomendado em função do comportamento das unidades de codificação. Além disso, tem a vantagem de ser facilmente adaptável a máquinas de qualquer comprimento de palavra.

Uma compactação maior dos dados pode normalmente ser obtida no uso combinado de técnicas aplicadas a uma mesma base de dados. No trabalho de Hahn ([6]) temos um exemplo de supressão de nulos, seguida pela codificação numérica dos caracteres não brancos.

REFERÊNCIAS BIBLIOGRÁFICAS

([1]) MYERS, W., TOWNSEND, M & TOWNSEND, T. Data compression by hardware or software. *Datamation, 11*: 39-43, 1966.

([2]) SEVERANCE, D.G. A practitioner's guide to data base compression. *Inform. Systems, 8* (1): 51-62, 1981.

([3]) FURTADO, A.L. & SANTOS, C.S. *Organização de Banco de dados*. Rio de Janeiro, Ed. Campos, 1979. Cap. 4.

([4]) MARTIN, J. *Computer data-base organization*. Englewood-Cliffs, N.J., Prentice-Hall, 1975.

([5]) SNYDERMAN, M. & HUNT, B. The myriad virtues of text compaction. *Datamation, 5*: 36-40, Dec. 1, 1970.

([6]) HAHN, B. A new technique for compression and storage of data. *Commun. ACM, 17* (8): 434-6, 1974.

([7]) CHEN, T.C. & HO, I. T. Storage-efficient representation of decimal data. *Commun. ACM, 18* (1): 49-52, 1975.

([8]) SMITH, A.J. Comments on a paper by T.C. Chen and I.T. Ho. *Commun ACM, 18* (8): 463, 1975.

([9]) DATE, C.J. *An introduction to database systems*. Reading, Mass., Addison-Wesley, 1982.

([10]) HEAPS, H.S. & THIEL, L.H. Program design for retrospective searches on large data base. *Inf. Storage Retr.*, 8 (1): 1-20, 1972.

*([11]) BLASGEN, M.W., CASEY, R.G. & ESWARAN, K.P. On encoding method for multifield sorting and indexing. *Commun. ACM, 20* (11): 874-8, 1977.

**([12]) RUBIN, F. Experiments in text file compression. *Commun. ACM, 19* (11): 617-23, 1976.

(*) O trabalho apresenta um projeto de codificação que produz uma cadeia única de caracteres, a partir de uma seqüência de cadeias de caracteres, mantendo o relacionamento de ordem. Este esquema visa à recuperação eficiente de registros em uma base de dados que satisfaçam uma expressão que envolva mais de um campo de dado.

A técnica apresentada não reduz comprimento de dado e, por outro lado, acrescenta redundância devido ao uso de caracteres de controle de ajuste de tamanho de campo.

O esquema é recomendado para aplicação que envolvam ordenamento ou indexação de campos múltiplos.

(**) O trabalho apresenta um sistema para a compressão de arquivos de textos. O sistema consiste em um programa de análise, um codificador e um decodificador.

O enfoque básico é dividir o texto em subcadeias de caracteres e substituir cada subcadeia por um código. São apresentados alguns algoritmos para realizar esta codificação.

As técnicas descritas no trabalho exigem grande tempo de computação e bastante memória. Seu uso é recomendado para textos que variam muito pouco ao longo do tempo.