Substituição Homofônica utilizando Códigos de Huffman Canônicos

José Rodrigues Fernandes (jose.rodrigues@ucp.br) Universidade Católica de Petrópolis (UCP)
Ruy Luiz Milidiú (milidiu@inf.puc-rio.br) Pontifícia Universidade Católica (PUC-Rio)
Claudio Gomes de Mello (cgmello@de9.ime.eb.br) Instituto Militar de Engenharia (IME)

Resumo. Neste trabalho, é proposto um algoritmo para compressão e criptografia simultâneas. O algoritmo utiliza um esquema de substituição homofônica. Os códigos para o conjunto de símbolos homofônicos são gerados via Códigos de Huffman Canônicos. Além disso, o algoritmo utiliza uma chave secreta para a fase final de cifragem baseada em permutações. Para testar a eficiência do algoritmo proposto, comparamos seu desempenho com um compressor convencional. Nos experimentos realizados, observamos uma perda de compressão inferior a 6%. Estes resultados indicam um bom compromisso entre qualidade e eficiência.

Palavras-chave. criptografia, substituição homofônica, compressão, Huffman canônico.

1. Introdução

Devido à explosão de utilização da Internet, o momento atual é de grandes desafios no gerenciamento e distribuição de documentos através de meio eletrônico. É crescente a demanda por serviços que necessitam da distribuição de informações de modo seguro e que tenham custos cada vez menores. Usualmente, estes documentos passam por dois processos distintos: compressão dos dados, que tem como objetivo principal a diminuição do custo de transmissão, e a cifragem, para evitar que pessoas não autorizadas tenham acesso à informação.

Neste artigo, é apresentado um algoritmo que realiza simultâneamente os processos de compressão e criptografia de dados. O algoritmo é baseado nos Códigos de Huffman Canônicos e na técnica criptográfica de Substituição Homofônica, além de usar uma chave secreta para a cifragem dos dados. O foco do algoritmo aqui proposto é o armazenamento e transmissão eletrônica de grandes coleções textuais. Os testes realizados mostram que a utilização da substituição homofônica não prejudica significativamente a compressão.

Na seção 2, é apresentado o código de Huffman e algumas de suas principais características. Na seção 3, são introduzidos os códigos de Huffman Canônicos, enquanto que na seção 4 a técnica de substituição homofônica é detalhada. Na seção 5, o algoritmo proposto é descrito e na seção 6 alguns testes realizados são mostrados. E, finalmente, na seção 7, tem-se as conclusões e trabalhos futuros.

2. Códigos de Huffman

Uma das melhores técnicas de compressão conhecidas é devida a Huffman [Huff52]. Para uma dada distribuição de probabilidades gera um código ótimo, livre de prefixo e de redundância mínima. Quando todas as probabilidades são potências negativas de dois além de produz uma sequência de bits aleatórios. Utiliza códigos de tamanho variável para representar os símbolos do texto, que podem ser caracteres ou cadeias de caracteres. A idéia básica do algoritmo é atribuir códigos de bits menores para os símbolos mais freqüentes no texto, e códigos mais longos para os mais raros. Assim, na codificação, consegue-se a compressão do arquivo em comparação a códigos de tamanho fixo (ex: ASCII).

O algoritmo original de Huffman gera um código ótimo com esforço computacional $O(n \log n)$. Gera uma árvore binária completa com n nós externos e $n-1$ nós internos, onde os nós externos (folhas) comportam os símbolos do texto, e as arestas são rotuladas com 0 (ramo da esquerda) ou 1 (ramo da direita). O código de cada símbolo é obtido percorrendo-se a árvore desde a raiz até a folha onde ele se encontra. É fácil verificar que o código é livre de prefixo, isto é, nenhum código é prefixo de outro. Consequentemente, nenhuma mensagem é decodificada de modo ambíguo.

Para decodificar basta percorrer a árvore a partir da raíz segundo a cadeia de bits da mensagem codificada até atingir uma folha que representa o símbolo decodificado. Então, o processo se repete.

Os códigos de Huffman que utilizam caracteres como os símbolos do alfabeto são chamados de Huffman de caracteres ou Huffman de bytes. Mas, para se conseguir melhores taxas de compressão, utiliza-se o Huffman de palavras, onde cada símbolo é uma cadeia de caracteres delimitada por separadores (vírgula, ponto, ponto-e-vírgula, etc.). Existe ainda o Huffman de n-gramas, onde os símbolos são n-uplas de caracteres.

O algoritmo de Huffman pode ser utilizado como um método de cifragem. Shannon [Shan49] sugeriu que a redundância da língua deveria ser reduzida antes da cifragem. O Código de Huffman possui redundância mínima dentre os códigos de prefixo.

Além disso, dado um arquivo codificado com Huffman, um tipo de ataque é a Busca Exaustiva [Klein89]. Dada uma cadeia S composta dos caracteres $\{c_1c_2c_3\ldots c_m\}$, Huffman gera uma cadeia de bits

$\{b_1b_2b_3\ldots b_n\}$ referentes aos códigos $\{d_1d_2d_3\ldots d_m\}$, onde d_i=código de Huffman de c_i. Então, uma busca exaustiva consiste em separar $\{b_1b_2b_3\ldots b_n\}$ em *m* partições não-vazias, o que significa testar uma combinação de (n-1)!/(n-1-m)!m! casos, que têm de ser verificados quanto às seguintes condições:

a) os códigos $\{d_1, d_2, d_3, \ldots\}$ distintos são códigos livres de prefixo;

b) a codificação é consistente, ou seja, se $c_i=c_j$ então tem-se $d_i=d_j$.

O tamanho exponencial de *m* e o custo $O(n^m)$ geralmente invalidam a busca exaustiva.

Mais ainda, Rivest et al. [Rive96] fizeram a criptoanálise de uma mensagem codificada com Huffman na qual o criptoanalista não conhece o modelo (alfabeto de símbolos). Demonstrou que a criptoanálise é extremamente difícil e até mesmo impossível em alguns casos, dado que a mensagem pode ser ambígua, ou seja, pode ter sido gerada por dois códigos de Huffman igualmente válidos.

Uma árvore de Huffman é uma árvore ótima, mas existem várias outras árvores ótimas. Algumas delas são obtidas facilmente trocando-se de posição os símbolos de mesmo nível na árvore. Isto pode ser usado para se embaralhar a codificação. Usando esta característica, Wayner [Wayn88] propôs um método no qual é feita uma operação de ou-exclusivo (XOR) entre a chave secreta e os rótulos das arestas da árvore de Huffman. Esta operação equivale a usar uma outra árvore.

Embora recentes publicações de métodos para compressão de dados deixem a impressão de que Huffman estaria obsoleto e superado pelas novas técnicas, ele ainda continua sendo muito útil principalmente para o armazenamento de grandes massas de textos estáticos, enquanto os novos métodos adaptativos, como o de Ziv and Lempel (utilizado nas ferramentas compress, pkzip, arj, etc.) e os Códigos Aritméticos, são preferíveis em algumas aplicações de tempo real e para comunicação de dados.

Os códigos de Huffman possuem uma série de vantagens, destacam-se [Klein93]: simplicidade; rapidez; tende a se auto-sincronizar após erros na transmissão; na busca, para encontrar uma cadeia S em um texto comprimido T, normalmente é necessário primeiro se descomprimir T e então procurar a cadeia, ao invés disso, como Huffman codifica cada símbolo sempre da mesma maneira, basta comprimir a cadeia S e então procurá-la em T; etc.

3. Códigos de Huffman Canônicos

Existe ainda uma outra representação do código de Huffman que é muito eficiente para a decodificação, mesmo que os modelos sejam extremamente grandes, que é o caso das grandes coleções textuais. Esta representação é chamada de códigos de Huffman Canônicos. São gerados através de um algoritmo que leva em consideração apenas o tamanho do código de Huffman para o símbolo, ou seja, a altura do símbolo na árvore.

Na Tabela 3.1, é exemplificada uma tabela com o código de Huffman e o respectivo código de Huffman Canônico, segundo o algoritmo descrito por Moffat et al [Moff94].

Dígito	Código de Huffman	Tamanho do código	Código de Huffman Canônico
0	00	2	10
1	10	2	11
2	110	3	011
3	1110	4	0101
4	0111	4	0100
5	0110	4	0011
6	11111	5	00001
7	0100	4	0001
8	1101	4	0010
9	11110	5	00000

Tabela 3.1 - Código de Huffman Canônico.

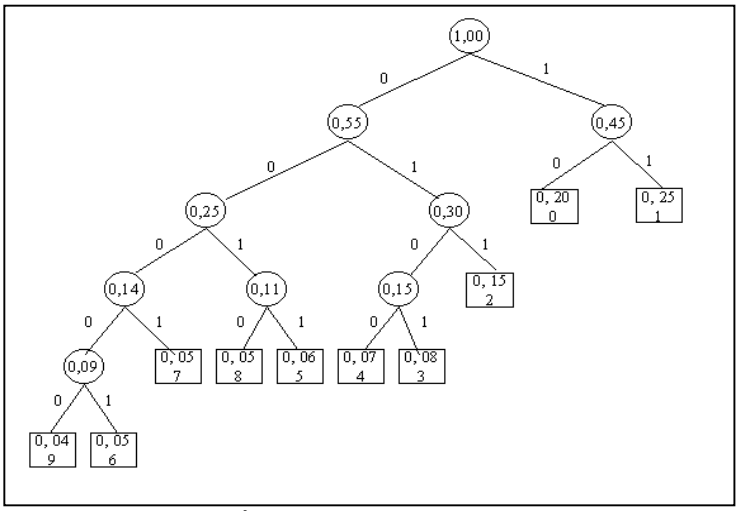

Figura 3.1 - Árvore do Código de Huffman Canônico

Na figura 3.1 está representada a árvore correspondente aos códigos de Huffman Canônicos da tabela 3.1 . Para construir a árvore, basta conhecer o tamanho do código de Huffman para cada símbolo, e a partir daí posicionam-se as folhas nos níveis adequados, da esquerda para a direita, ordenados ascendentemente pela freqüência. Depois, completa-se a árvore criando os nós internos. O algoritmo de Huffman tradicional pode ser utilizado para determinar os tamanhos dos códigos. O processo de construção dos códigos de Huffman Canônicos é rápido e econômico. Não é necessária a construção de uma árvore realmente, o algoritmo utiliza

apenas o comprimento do código de Huffman para derivar o código Canônico, e a grande vantagem é um ganho substancial no processo de decodificação.

4. Substituição Homofônica de Tamanho Variável

A história da criptografia demonstra que um outro tipo de ataque muito utilizado é o Ataque Estatístico, onde um conhecimento das particularidades de uma determinada língua é usado heurísticamente para a quebra dos códigos. Uma defesa contra este tipo de ataque é a técnica de substituição homofônica.

A substituição homofônica consegue suavizar a distribuição de probabilidades (ou frequências) dos símbolos de um texto em claro. Desta forma, a tarefa do criptoanalista é dificultada. A substituição homofônica convencional associa um símbolo a um conjunto de símbolos substitutos possíveis, chamados homofônicos, tais que a distribuição destes novos símbolos seja equiprovável. Assim, cada símbolo do alfabeto original é desdobrado em outros equiprováveis.

Em 1988, Günther [Gunt88] introduziu uma importante generalização da substituição homofônica. Neste caso, uma nova forma de escolha dos homofônicos é especificada. Um ano mais tarde, Massey et al. [Mass89] aplicaram conceitos de Teoria da Informação à proposta de Günther e formularam a "Substituição Homofônica de Tamanho Variável". Este novo tipo de substituição homofônica é uma generalização da convencional, onde tem-se homofônicos representados por códigos de comprimentos diferentes, e ainda, cada símbolo do alfabeto original pode ser substituído por homofônicos não equiprováveis, ou seja, os homofônicos poderão ter probabilidades distintas. Este é o objetivo da substituição homofônica, converter a seqüência de texto em claro em uma seqüência completamente aleatória.

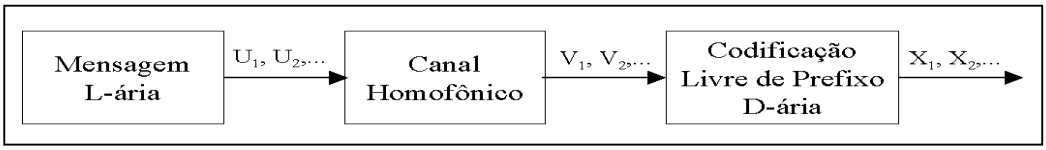

Figura 4.1 - Um esquema geral para substituição homofônica.

A Figura 4.1 representa um esquema de substituição genérico para representar tanto a **substituição homofônica convencional** como a **substituição homofônica de tamanho de variável**. O canal homofônico é sem memória. O alfabeto de entrada { u_1, u_2, ..., u_L } coincide com o grupo de símbolos possíveis de U. O alfabeto de saída { v_1, v_2, v_3, ... } também é finito ou infinitamente contável, e com probabilidades de transição $P(V=v_j \mid U=u_i)$. Para cada *j* existe exatamente um *i* tal que $P(V=v_j \mid U=u_i) \neq 0$. Cada v_j com $P(V=v_j \mid U=u_i) > 0$ será um homofônico para u_i.

Quando o canal homofônico da Figura 4.1 é determinístico, todas as probabilidades de transição diferentes de zero são iguais a 1 (podemos também dizer que V=U). Neste caso, então a Figura 4.1 representa uma codificação de fonte usual (ou até mesmo uma compressão de dados).

Quando o canal homofônico é não determinístico mas a codificação binária livre de prefixo é simples, ou seja, todos os códigos têm o mesmo comprimento *m*, então a Figura 4.1 representa substituição homofônica convencional.

Quando o canal homofônico é não determinístico e a codificação binária livre de prefixo possui comprimento variável, então a Figura 4.1 representa **substituição homofônica de tamanho de variável**, como definida por Günther [Gunt88].

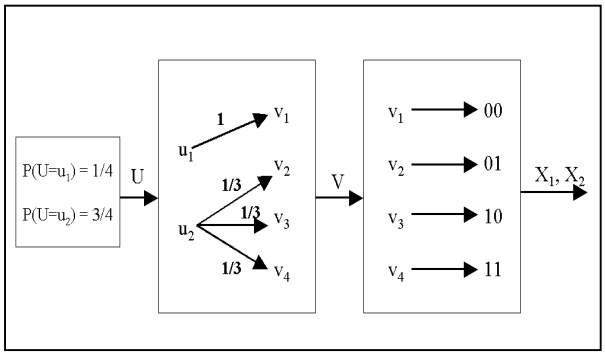

Figura 4.2 - Substituição homofônica convencional.

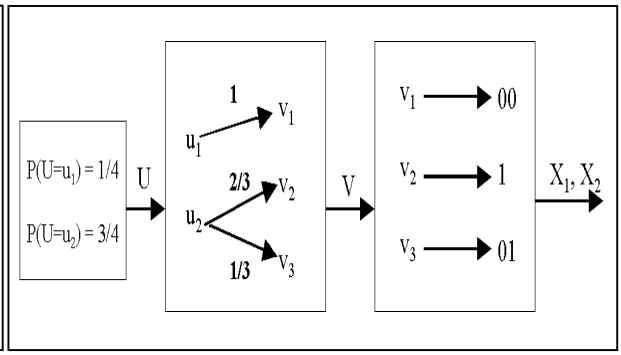

Figura 4.3 - Substituição homofônica de tamanho variável.

As Figuras 4.2 e 4.3 mostram dois exemplos do esquema geral de substituição homofônica ilustrado na Figura 4.1, ambos para a mesma mensagem de fonte binária (isto é, L = 2). A substituição de tamanho variável tem um comprimento médio do código E[W] = 3/2 e a substituição convencional tem E[W] = 2. Esta redução dos símbolos codificados é uma vantagem da substituição homofônica de tamanho de variável perfeita.

Uma substituição homofônica é perfeita se a seqüência D-ária codificada como X_1, X_2, ... é completamente aleatória. Para o caso sem memória (fonte e canal), considerado na Figura 4.1, isto é equivalente à condição que o código X_1, X_2, ... , X_w para $V = V_1$ é completamente aleatório.

Uma substituição homofônica D-ária é ótima para uma dada mensagem fonte, se e somente se é perfeita e minimiza o comprimento médio do código E[W].

Para simplificar os conceitos, foi utilizado nesta seção D = 2, ou seja, representação binária. Massey et al. [Mass89] apresentam a generalização correspondente.

Para cada símbolo *u* de U, as probabilidades dos homofônicos de *u* formam uma decomposição de P(U = *u*) em uma soma dos números referentes as potências inteiras negativas de 2. Por exemplo, na Figura 4.3 P(U = u_2) = 3/4 decompõem-se como 1/4 + 1/4 + 1/4 e 1/2 + 1/4, respectivamente.

Massey et al. [Mass89] mostram que se o canal é ótimo então a decomposição de P (U = u) para qualquer u deve consistir em diferentes potências negativas de 2. Se $P(U = u) = i/2^n$ para alguns números inteiros *i* e *n*, então pode ser decomposta em uma soma finita de distintas potências negativas de 2. Caso seja diferente, então teremos uma soma infinita de distintas potências negativas de 2.

Demonstram também um limite superior estreito de H(V) para um canal homofônico ótimo:

$$H(U) \leq H(V) = E(W) < H(U) + 2.$$

Esta conclusão mostra que um canal homofônico ótimo nunca aumenta a entropia de sua entrada U por mais de dois bits, independente do valor de H(U).

5. Descrição do Algoritmo

A) O primeiro passo do algoritmo é a varredura do texto (parsing) para acumular as freqüências (probabilidades), de cada símbolo do alfabeto no texto em claro. Os símbolos são armazenados em uma árvore binária.

B) Então, uma substituição homofônica de tamanho variável é utilizada. Os homofônicos de cada símbolo serão representados pelos termos da decomposição em potências negativas de 2 do valor da sua probabilidade. Assim é conseguida uma distribuição diádica, ou seja, todas as probabilidades serão potências negativas de 2. Por exemplo, para o símbolo com probalidade igual a 3/4, tem-se dois símbolos homofônicos: um com probabilidade igual a 2^{-1} e o outro igual a 2^{-2}. Como conseqüência deste fato, temos que o número total de símbolos pode crescer muito, tanto quanto o número de termos da decomposição, o que pode resultar ainda em dízimas periódicas. Uma solução simples é o truncamento. Por exemplo, neste algoritmo o número máximo de termos é limitado a 8.

C) Com o objetivo de obter uma decodificação rápida, foi utilizado o algoritmo de codificação via Códigos de Huffman Canônicos para determinar os códigos de cada símbolo homofônico. Este algoritmo necessita apenas do tamanho do código para realizar o processamento. O caminho natural seria utilizar o algoritmo de Huffman tradicional para determinar os tamanhos dos códigos dos símbolos homofônicos.

Mas, como cada um dos símbolos possui probabilidade igual a uma potência negativa de 2 e a soma de todas as probabilidades é igual a 1, o algoritmo de Huffman tradicional fornece o tamanho do código igual ao expoente. Por exemplo, um símbolo com probalidade 2^{-5} terá tamanho de código igual a 5. Então, o algoritmo de Huffman Canônico utiliza essa propriedade para gerar diretamente os códigos canônicos, mesmo considerando que soma de todas as probabilidades pode não ser igual a 1, devido ao truncamento na decomposição.

No momento da codificação, quando um símbolo X for codificado, na verdade um de seus homofônicos {X_1, X_3, ..., X_8} será o escolhido. Há duas alternativas para a escolha: seleção sequencial, onde na primeira vez será usado o X_1, na segunda o X_2, até que após o X_8 inicia-se novamente em X_1 e assim por diante; ou a seleção aleatória, definida por Massey et al. [Mass89] e que foi utilizada neste trabalho, onde um dos homofônicos é sorteado independentemente da escolha anterior.

D) O algoritmo utiliza uma chave secreta de tamanho variável (256/512/1024 bits, etc.) para a cifragem na codificação. Na verdade é uma chave de sessão (s-key) trocada anteriormente através de um meio considerado seguro, ou através de protocolos de troca de chaves como os que utilizam esquemas de chave pública.

A chave define permutações sobre os símbolos que pertencem a um mesmo nível da árvore canônica. Assim, é feita uma permutação inicial sobre os símbolos de mesmo tamanho de código como exemplificado na figura 5.1.

Suponha um alfabeto de símbolos {A, B, C, D, E, F} e os conjuntos de homofônicos definidos por decomposição das probabilidades como sendo os conjuntos {A_1, A_2}, {B_1, B_2, B_3}, {C_1, C_2, C_3}, {D_1, D_2}, {E_1,

E_2, E_3, E_4, E_5, E_6, E_7, E_8}e {F_1, F_2, F_3, F_4, F_5}, respectivamente. Então, supondo que A_1, B_3, C_2, D_1, E_3 e F_4 possuam a mesma probabilidade igual a 2^{-x}, eles ocuparão o mesmo nível na árvore canônica.

A cifragem irá consistir na permutação inicial destes símbolos de mesmo nível a partir da ordem definida pela chave secreta agrupada em bytes. Para uma chave de 1024 bits, tem-se 1024/8=128 elementos a permutar, o que possibilita 128! permutações. Caso um nível possua mais do que 128 símbolos, basta agrupá-los em blocos e aplicar as permutações a cada bloco. Abordagens alternativas para o caso de níveis com mais de 128 símbolos podem ser: ciclar a chave ou expandir a chave a partir de funções pseudo-aleatórias.

E) Quando o símbolo A_1 for codificado, na verdade o código que será gravado no arquivo codificado será o do símbolo C_2. O símbolo B_3 será substituído pelo E_3 e assim por diante. O que esse esquema propõe é que quando o símbolo A for codificado, e o seu homofônico A_1 for o escolhido, na verdade o C_2 é que será codificado.

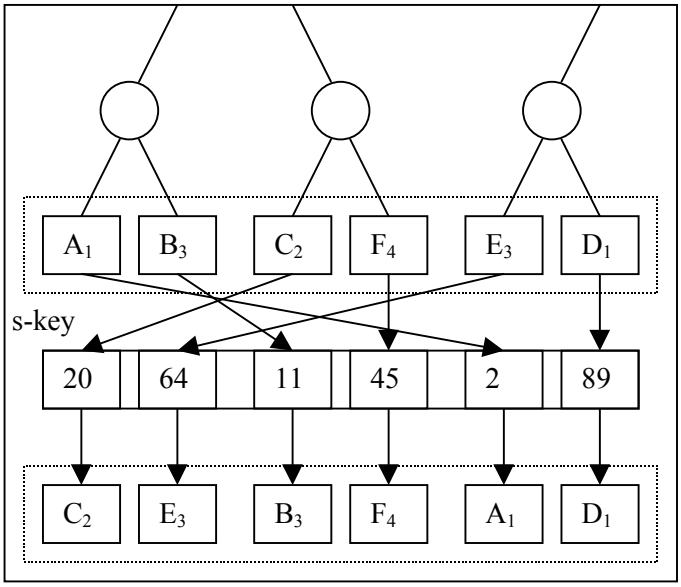

Figura 5.1 – Esquema de cifragem com permutações.

Adicionalmente, uma operação Troca é aplicada fazendo com que o símbolo usado na codificação seja realocado para a primeira posição no nível. Por exemplo, quando o símbolo B_3 for codificado, ele fará com que o símbolo E_3 troque de lugar com o símbolo que ocupa a primeira posição. Assim, a cada codificação tem-se uma nova configuração e cada símbolo terá diferentes codificações a cada iteração, devido à escolha do homofônico que irá representá-lo e/ou devido à operação Troca.

Para conservar a segurança do código de Huffman demonstrada por Rivest et al. [Rive96] é necessário que o modelo não seja acessível ao criptoanalista. A solução é aplicar o próprio algoritmo também ao modelo. Além disso, o algoritmo foi analisado quanto aos tipos de ataques básicos [Stin95] e mostrou-se seguro. Os ataques diferem quanto ao nível de acesso que o criptoanalista possui, sendo que o objetivo é descobrir a senha secreta utilizada: o inimigo possui apenas um arquivo codificado; o inimigo possui um arquivo X e o arquivo codificado correspondente Y; o inimigo tem acesso ao algoritmo de codificação (com a chave embutida) e pode entrar com X para receber Y; e, no mais alto nível de acesso, o inimigo usa o algoritmo de decodificação (com a chave embutida) com um arquivo codificado Y escolhido e obtém o decodificado X.

Considerando a descrição do algoritmo de codificação, foi possível desenvolve-lo de forma modular. Resumindo, os módulos são os seguintes:

A) **Parser;**
B) **Determinar os símbolos homofônicos por decomposição;**
C) **Determinar os códigos dos símbolos homofônicos via Códigos de Huffman Canônicos;**
D) **Determinar a permutação inicial para a codificação utilizando a chave secreta;**
E) **Codificar o do texto com operações Troca e sorteio para escolha dos homofônicos.**

Com o objetivo de minimizar o custo computacional no momento da distribuição do documento, este procedimento pode também ser executado em duas etapas. Para um determinado documento, primeiro podem ser executadas as fases A, B e C, e o resultado fica armazenado. No ato da distribuição, de posse da chave secreta, é realizado o processamento das fases D e E.

O procedimento de decodificação utiliza o algoritmo dos Códigos de Huffman Canônicos, associado à chave secreta do usuário para recriar as permutações.

6. Experimentos

Na tabela 6.1 os testes realizados são listados. A taxa de compressão é a razão entre o tamanho do texto comprimido e o tamanho do texto original. O algoritmo aqui proposto foi implementado em C utilizando a ferramenta Visual C++ 6.0, em ambiente Windows 98.

		Huffman Canônico		Homofônico Canônico	
Textos (*)	**Tamanho Original (bytes)**	**Tamanho Codificado (bytes)**	**Taxa de Compressão (%)**	**Tamanho Codificado (bytes)**	**Taxa de Compressão (%)**
(1) José de Alencar	2.843.945	1.349.914	47,47%	1.511.191	53,14%
(2) Aluísio de Azevedo	1.911.875	1.027.953	53,77%	1.106.489	57,87%
(3) 0drvb10.txt	5.910.467	2.141.261	36,23%	2.477.910	41,92%
(4) csnva10.txt	6.872.548	2.496.646	36,33%	2.899.392	42,19%
(5) kjv10.txt	4.445.260	1.584.097	35,64%	1.843.699	41,48%
(6) taofj10.txt	3.008.331	1.069.052	35,54%	1.236.673	41,11%

(*) Os textos 1 e 2 são coleções de livros de autores brasileiros, e os textos 3, 4, 5 e 6 foram retirados da coleção do Projeto Gutemberg.

Obs.: As taxas de compressão dos textos em português observadas foram significativamente maiores provavelmente devido ao maior número de palavras possíveis (modelo).

Tabela 6.1 – Comparação entre Huffman Canônico e Homofônico Canônico

7. Conclusão

O objetivo deste trabalho é desenvolver um algoritmo que execute a compressão e a cifragem de texto simultaneamente. O foco atual é o gerenciamento eletrônico de grandes coleções textuais e grandes documentos. Neste caso, o tamanho do modelo de palavras é relativamente pequeno quando comparado ao do texto. Além disso, o parsing do texto para a definição das freqüências exatas é um passo que tem um custo muito grande, e que também não é indicado para documentos muito pequenos, nesse caso os algoritmos adaptativos ou mesmo o Huffman dinâmico pode ser mais indicado.

A tabela 6.1 mostra que a perda máxima de compressão devido ao uso da substituição homofônica ficou em 5,86%. O algoritmo ainda está em desenvolvimento e algumas melhorias já estão previstas. Uma forma de tratar o crescimento exagerado do modelo é utilizar técnicas de redução como definida por Zobel et al. [Zobel99] em que apenas os símbolos com freqüências significativas, ou acima de um limite mínimo, são codificados com Huffman. Os símbolos mais raros que teriam códigos muito grandes são substituídos por um caractere indicativo (caractere de escape) seguido dos códigos ASCII dos caracteres que compõem a cadeia da palavra. Para melhorar a rapidez do algoritmo algumas mudanças também estão sendo estudadas como a otimização do código em C e uso de estruturas de dados alternativas como árvores AVL para o parsing.

Algumas medições têm de ser feitas para a comparação com soluções serializadas do problema. Aliado a isso, estão sendo melhor analisadas a segurança atual do algoritmo, bem como a introdução de novas estruturas para incrementá-la, como por exemplo redes de Feistel ou o uso de S-Boxes. Então, novos testes serão feitos assim como a criptoanálise do novo esquema.

Bibliografia

[Gunt88] Gunter, C.G. 1988. *An Universal Algorithm for Homophonic Coding.* in Advances in Cryptology Eurocrypt-88, LNCS, vol. 330.

[Huff52] Huffman, D. 1952. *A Method for the Construction of Minimum-Redundancy Codes*. Proc. IRE, 1098-1101.

[Klein89] Klein, S. T., Bookstein, A., Deerwester, S. 1989. *Storing Text Retrieval Systems on CD-ROM: Compression and Encryption Considerations*. ACM Trans. on Information Systems, vol. 7, no. 3

[Klein93] Bookstein, A., Klein, S.T. 1993. *Is Huffman coding dead ?*, Computing 50 279--296

[Mass89] Massey, J.L., Kuhn, Y.J.B., Jendal, H.N. 1989. *An Info*rmation-*Theoretic Treatment of Homophonic Substitution*. in Advances in Cryptology Eurocrypt-89, LNCS, vol. 434.

[Moff94] Moffat, A., Witten, I.I., Bell, Timothy C. 1994. *Managing Gigabytes: Compressing and Indexing Documents and Images.* Van Nostrand Reinhold.

[Rive96] Rivest, Ronald L., Mohtashemi, M., Gillman, David W. 1996. *On Breaking a Huffman Code*. IEEE Transactions on Information Theory, vol. 42, no. 3.

[Shan49] Shannon, C. 1949. *Communication Theory of Secrecy Systems*. Bell Syst. Tech., vol. 28, no. 4, pp. 656-715.

[Stin95] Stinson, Douglas S. 1995. *Cryptography - Theory and Practice*. CRC Press.

[Wayn88] Wayner, P. 1988. *A Redundancy Reducing Cipher*, Cryptologia, 107-112.

[Zobel99] Zobel, J., Williams, H.E. 1999. *Compact In-Memory Models for Compression of Large Text Databases*. SPIRE´99 (String Processing and Information Retrieval), 224-231