SISTEMA ADAPTATIVO PARA COMPRESSÃO DE DADOS

NEWTON FALLER

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIA (M. Sc.)

Aprovada por:

Ysmar Vianna e Silva Filho
Prof. Ysmar Vianna e Silva Filho
Presidente

Ivan da Costa Marques
Prof. Ivan da Costa Marques

Nelson Maculan Filho
Prof. Nelson Maculan Filho

Prof. Sergio Teixeira

RIO DE JANEIRO
ESTADO DA GUANABARA - BRASIL
DEZEMBRO DE 1973

DEDICATÓRIA

A MEUS PAIS

ADA E KURT

AGRADECIMENTO

Aos Professores JOHN HEINBOLD e CARLOS HARTMANN por algumas idéias iniciais.

Ao Professor YSMAR VIANNA pela orientação durante todo desenvolvimento do trabalho.

Ao Professor JOÃO LIZARDO pelas sugestões relativas ao teste do sistema.

A MARIA ALICE F. C. MELLO pela extrema dedicação e paciência na preparação datilografada dos originais.

RESUMO

Uma propriedade interessante é provada para árvore de Huffman.
Quaisquer dois elementos de pesos a_i e b_i filhos de um mesmo pai tem a seguinte propriedade :

$$\text{se} \quad b_i \geqslant a_i \qquad \text{então}$$

$$W_m \leqslant a_i \qquad \text{ou} \qquad W_m \geqslant b_i$$

onde W_m é o peso de qualquer nó da árvore.

Baseado nesta propriedade, um algoritmo foi desenvolvido para atualizar dinâmicamente uma árvore de Huffman, à medida que os pesos dos seus nós terminais variam.

Utilizando-se este algoritmo, um modelo de um sistema adaptativo para compressão de dados foi implementado.

Simulações efetuadas com diversos tipos de dados levaram a resultados interessantes.

ABSTRACT

An interesting property is proven for Huffman's tree. Any two elements with weights a_i and b_i sons of a same father have the following property :

If $b_i \geqslant a_i$ then

$W_m \leqslant a_i$ or $W_m \geqslant b_i$

where W_m is the weight of any node of the tree.

Based on this property, an algorithm is developed to dynamically update Huffman's tree as weights of terminal nodes change.

Using this algorithm, a model of an adaptive system for data compression is developed.

Simulation using many types of data led to interesting results.

ÍNDICE

I. INTRODUÇÃO

Uma técnica que se tem procurado sempre aprimorar é a compressão de dados. Comprimir dados significa reduzir a redundância inerente aos conjuntos de dados a serem codificados, tendo como objetivo representar apenas a informação real contida em tal conjunto.

Nos sistemas digitais a informação é codificada através de uma sequência de digitos binários (bits). Isto se deve ao fato de que na tecnologia atual é mais fácil reconhecer apenas dois estados (0 e 1) em um dispositivo do sistema.

Dos estudos de Teoria da Informação [1] sabe-se que a entropia de uma fonte binária ergódica pode ser definida como

$$H = - \sum_{n=1}^{n} p_i \log_2 p_i \quad \text{bits/símbolo}$$

onde n é o número de símbolos a ser codificado e p_i é a probabilidade de ocorrência do i-ésimo símbolo do conjunto de dados.

Informalmente, a entropia de uma fonte define o número mínimo de bits por símbolo necessários para codificar mensagens geradas por esta fonte. Este número, entretanto, é um limite inferior e em geral não é inteiro. Como o número de bits que representa determinado símbolo tem que ser um número inteiro, sòmente utilizando-se códigos de tamanho variável é possível aproximar-se deste limite inferior. Um método elegante para definir estes códigos, sabendo-se de antemão a probabilidade de ocorrência de cada um dos símbolos, foi dado por Huffman [2] .

Um código binário pode ser representado por uma árvore binária em que os caminhos à esquerda de cada nó correspondem aos zeros enquanto os à direita correspondem aos uns. A informação é associada aos nós terminais. Achar o símbolo correspondente a determinada codificação, significa partir da raiz da árvore seguindo à direita ou à esquerda, conforme os bits 1 ou 0 da codificação até alcançar-se um nó terminal. Existe portanto um caminho único que une a raiz da árvore a um nó terminal.

Neste trabalho considerou-se uma árvore de Huffman na qual os pesos associados aos diversos nós terminais variavam com o tempo. Um algoritmo para atualizar a árvore de forma a mantê-la sempre numa árvore de Huffman, foi desenvolvido. Baseado neste algoritmo um modelo de um sistema auto-adaptativo foi implementado.

Usando este sistema, não há necessidade de se saber a priori a frequência de ocorrência de cada símbolo do conjunto. À medida que um símbolo é codificado, o sistema computa a sua frequência relativa e adapta-se de tal maneira a manter um código com mínima redundância.

2. DEFINIÇÕES

A não ser nos pontos onde for explicitamente declarado, todas as definições e notações referentes a árvores são as mesmas usadas pelo Knuth [3] .

Árvore binária (chamada árvore binária extendida pelo Knuth) considera-se um nó chamado raiz com as suas duas árvores binárias distintas.

Floresta é um conjunto de árvores.

Nó interno, incluindo a raiz, tem dois filhos, o direito e o esquerdo, que são as raizes das sub-árvores direita e esquerda, correspondentes ao nó pai.

Nós terminais, nós externos ou folhas são nós que não possuem filhos.

Ao longo deste trabalho supõe-se sempre n nós terminais. O termo nó pode designar indistintamente um nó interno ou externo. Existe um único caminho entre a raiz e cada um dos nós (interno ou externo).

Comprimento do caminho de um nó é o comprimento do caminho (número de arcos) percorrido entre a raiz e este nó.

Comprimento do caminho de uma árvore é a soma dos comprimentos do caminho de todos os nós terminais. Assim, se l_j é o comprimento do caminho do j-ésimo nó terminal, então $\sum l_j$ é o comprimento do caminho da árvore.

Um nó está no nível k quando o comprimento do caminho deste nó é igual a k. A raiz está no nível zero e é considerado o mais alto nível.

Em muitos problemas associa-se a cada nó terminal um peso w. Comprimento ponderado do caminho de uma árvore ou simplesmente custo de uma árvore define-se como $\sum w_j l_j$ onde w_j e l_j são respectivamente o peso e o comprimento do caminho do j-ésimo nó terminal. Comprimento ponderado médio do caminho de uma árvore ou custo normalizado de uma árvore é dado por $\sum w_j l_j / \sum w_j$.

A árvore de menor custo possível de se construir com os mesmos n nós terminais chama-se árvore ótima.

Lema 2.1. : O número total de nós internos numa árvore binária é um menos que o número total de nós terminais [3] . Associemos também a cada nó interno um peso que seja a soma dos pesos de seus nós filhos.

A árvore construída com estes pesos associados a cada um de seus nós é a árvore que será considerada deste ponto em diante. Os pesos dos nós internos serão designados por w'.

Lema 2.2. : A soma dos n-1 pesos dos n-1 nós internos é o comprimento ponderado da árvore ou o custo da árvore [3] .

Assim $\sum w_i' = \sum w_j l_j$.

3. A ÁRVORE DE HUFFMAN

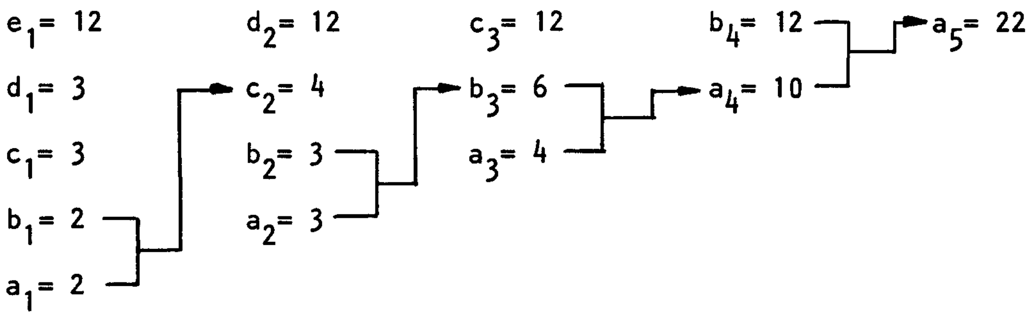
A árvore de Huffman é aquela construida através do algoritmo de Huffman 2 . O algoritmo de Huffman é o seguinte :

1) Colocar os n pesos dos nós terminais em ordem ascendente ;

2) Tirar os dois de menor valor da sequência, somá-los e introduzir esta soma novamente na sequência de forma a manter a ordem ascendente. Associar os dois valores retirados da sequência a nós filhos de uma sub-árvore cuja raiz tem peso igual à soma destes valores.

3) Repetir o passo 2 até que exista apenas um valor na sequência. Este valor é então associado à raiz da árvore.

(a) ALGORITMO DE HUFFMAN



(b) ÁRVORE DE HUFFMAN

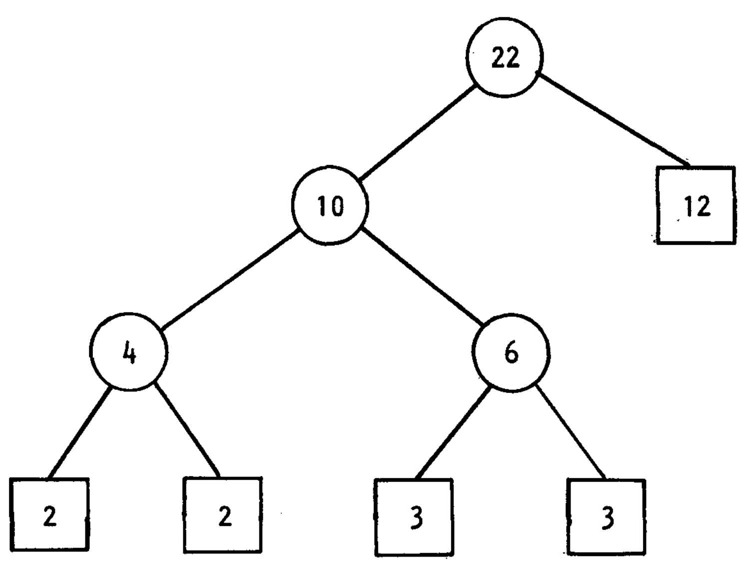


FIGURA 1. EXEMPLO DO ALGORITMO DE HUFFMAN E CONSTRUÇÃO DA ÁRVORE CORRESPONDENTE.

O algoritmo de Huffman constroi uma árvore ótima a partir de um conjunto de n pesos dados [2].

Suponha que paremos o algoritmo de Huffman após m passos executados. ($1 \leqslant m \leqslant n-1$). As sub-árvores já formadas são também árvores ótimas relativas aos valores dos pesos de seus nós terminais.

Lema 3.1. : O algoritmo de Huffman constrói uma floresta ótima de m somas [4]. Portanto, toda sub-árvore de uma árvore de Huffman é também uma árvore de Huffman. (Vide Figura 1).

4. ORDEM ASCENDENTE NA ÁRVORE DE HUFFMAN

<u>Lema 4.1.</u> : Seja <u>p</u> o peso associado à raiz da sub-árvore T_p no nível <u>k</u>. A contribuição de T_p para o custo total da árvore é dada por

$$C_1 = C_{T_p} + p.k \quad , \text{ onde } \quad C_{T_p} = \sum_{j \in T_p} w_j' \text{ é o custo de } T_p .$$

<u>Prova</u> : Substituindo-se T_p por um nó com peso <u>zero</u>, a soma dos nós internos decresce por duas razões :

1. Os nós internos de T_p não contribuem mais para a soma total.

2. A contribuição do peso <u>p</u> nos <u>k</u> nós internos no caminho até a raiz das árvores, desaparece.

Como pelo Lema 2.1. o custo de uma árvore é dado por $C = \sum w_j'$ temos que o novo custo da árvore é :

$$C' = C - \sum_{j \in T_p} w_j' - p.k \text{ , onde } \sum_{j \in T_p} w_j' = C_{T_p} \quad \text{, custo de } T_p .$$

<u>Lema 4.2.</u> : Seja <u>p</u> e <u>q</u> os pesos associados à raiz da sub-árvore T_p no nível <u>k</u> e a raiz da sub-árvore T_q no nível <u>h</u>. Se a árvore é ótima e $p \geqslant q$ então $k \leqslant h$.

<u>Prova</u> : Inverter as posições das sub-árvores T_p e T_q. Pelo Lema 4.2. o novo custo da árvore é

$$C' = C \, . \, (C_{T_p} + p.k) + (C_{T_q} + p.h) - (C_{T_q} + q.h) + (C_{T_p} + q.k)$$

$$C' = C + (q - p)\ (k - h).$$

Se a árvore original era ótima e nenhum nó terminal foi inserido ou retirado, então a troca de T_p por T_q não pode diminuir o custo da árvore.

Assim $(q - p)\ (k - h) \geqslant 0$ e

se $p \geqslant q$ então $k \leqslant h$.

Portanto em uma árvore ótima os nós com pesos maiores situam-se em níveis mais altos.

4.1. TEOREMA

Seja a_i e b_i os pesos associados aos nós filhos de um mesmo pai em uma árvore binária com $a_i \leqslant b_i$. A árvore é uma árvore de Huffman se e sòmente se $w_m \leqslant a_i$ ou $w_m \geqslant b_i$ onde w_m é o peso de qualquer nó da árvore.

4.1.1. Primeira Parte

Se a árvore é de Huffman então

$$w_m \leqslant a_i \quad \text{ou} \quad w_m \geqslant b_i \qquad \text{para todo } w_m$$

Prova : Suponha no passo i do algoritmo de Huffman os pesos : a_i , b_i , c_i , d_i , etc. onde $a_i \leqslant b_i \leqslant c_i \leqslant d_i$ etc. Assim, a_i e b_i são os dois menores valores dos pesos no passo i, como a_j e b_j o são no passo j. Processando este passo, três casos podem ocorrer :

1) $a_i + b_i \leqslant c_i$

2) $c_i < a_i + b_i \leqslant d_i$

3) $d_i < a_i + b_i$

Caso 1) : $b_{i+1} = c_i \geqslant a_i + b_i = a_{i+1} \geqslant b_i \geqslant a_i$

Caso 2) : $b_{i+1} = a_i + b_i > c_i = a_{i+1} \geqslant b_i \geqslant a_i$

Caso 3) : $b_{i+1} = d_i \geqslant c_i = a_{i+1} \geqslant b_i \geqslant a_i$

Portanto, para qualquer um dos casos a desigualdade $b_{i+1} \geqslant a_{i+1} \geqslant b_i \geqslant a_i$ é verificada. Desta forma os pesos que estão sendo processados são sempre maiores ou iguais àqueles já processados. Na sequência os pesos sendo processados são sempre os dois menores. O peso produzido sempre é maior ou igual àqueles que o produzem.

Portanto, a seguinte sequência pode ser construída :

$$a_1 \leqslant b_1 \leqslant a_2 \leqslant b_2 \leqslant \cdots \leqslant a_i \leqslant b_i \leqslant \cdots \leqslant a_n.$$

Assim, os valores de a_i e b_i são contiguos. Como a_i e b_i são os pesos associados aos nós filhos de um mesmo pai, o teorema está provado.

4.1.2. Segunda Parte

Se $w_m \leqslant a_i$ ou $w_m \geqslant b_i$, então a árvore é uma árvore de Huff - man.

Prova : Construir as duas sequencias baseadas na árvore :

1. $w_i \leqslant w_2 \leqslant w_3 \leqslant \cdots$ para os <u>n</u> nós terminais.

2. $w_i' \leqslant w_2' \leqslant w_3' \leqslant \cdots$ para os <u>n-1</u> nós internos.

Pela hipótese os pesos w_i' correspondem à soma de dois pesos contíguos.

Aplica-se o algoritmo de Huffman à sequência 1.

No primeiro passo retiram-se os pesos w_1 e w_2.

Como w_1' é a menor soma de dois outros pesos, necessàriamente $w_1' \geqslant w_2 \geqslant w_1$, pois w_1 e w_2 são os dois menores pesos correspondendo a nós terminais e todos outros pesos correspondendo a nós internos são maiores ou iguais a w_1'. Assim, w_1 e w_2 são contíguos e por esta razão a soma deles e xiste na sequência 2. Na realidade a soma deles é igual a w_1', dado que os dois menores valores devem produzir a menor soma. Retira-se o valor w_1' da sequência 2 e insere-se no lugar apropriado da sequência 1. Como $w_1' \geqslant w_2 \geqslant w_1$, a inserção de w_1' não quebra a contigüidade de w_1 e w_2 na sequência geral.

No passo <u>i</u> o procedimento é análogo. Neste caso os dois menores valores na sequência 1 podem ser valores correspondendo a nós internos ou externos e w_i' toma o lugar de w_1'.

Após <u>n-1</u> passos, o valor w_{n-1}', o peso da raiz da árvore é obtido. Como é possível reconstruir a árvore original usando o algoritmo de Huffman, a árvore original é uma árvore de Huffman.

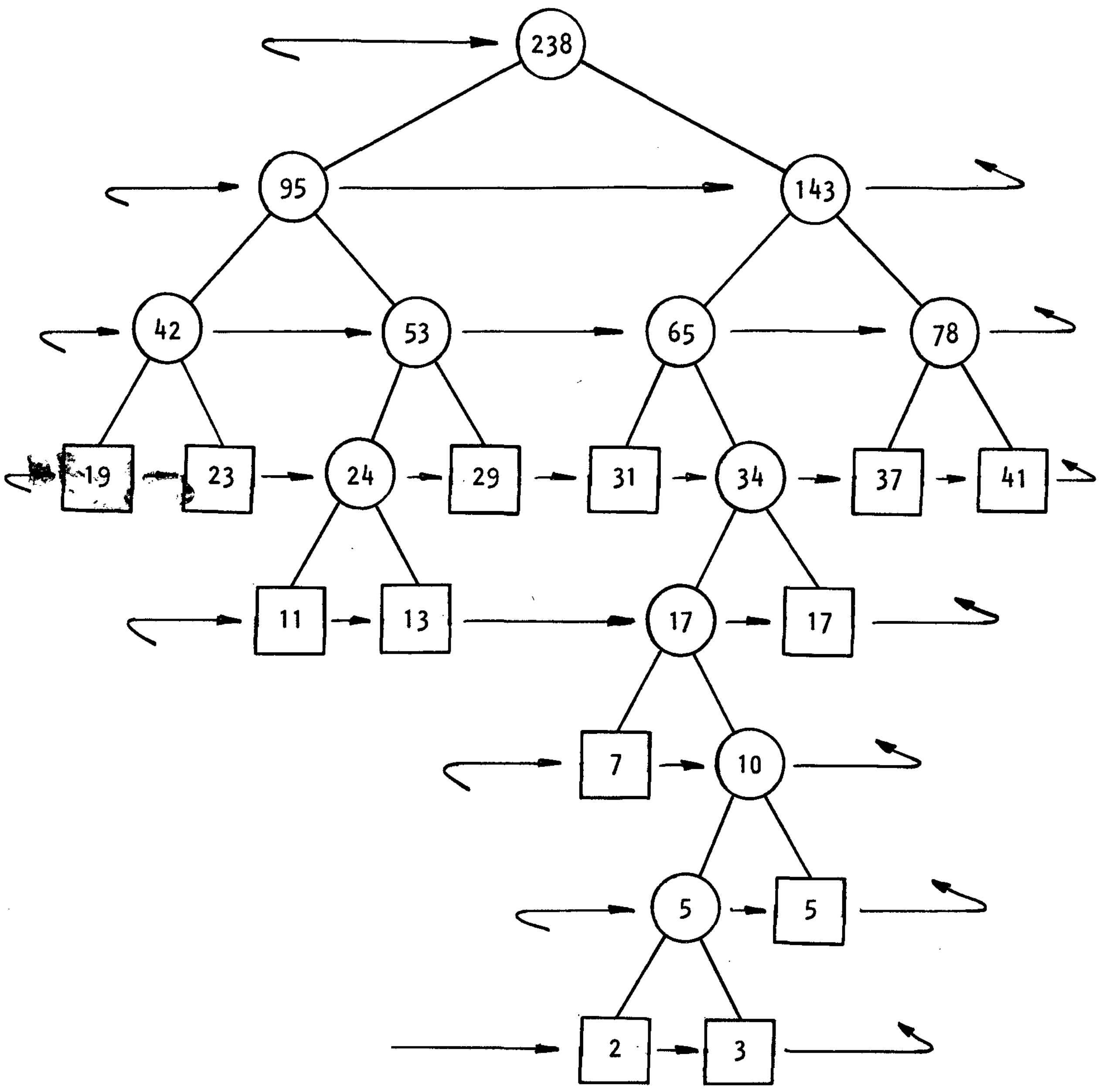


FIGURA 2. ÁRVORE DE HUFFMAN TIRADA DO KNUTH (3) PÁG. 403 , LIGEIRAMENTE MODIFICADA PARA MOSTRAR A PROPRIEDADE.

5. O ALGORITMO ADAPTATIVO

5.1. Introdução

Numa árvore binária os símbolos a serem codificados são associados aos nós terminais. Codificar um símbolo corresponde a achar o caminho que liga a raiz da árvore a este nó particular. Para atualizar os pesos dos nós à medida que os símbolos vão sendo codificados , é necessário que se incremente de uma unidade os pesos de todos os nós pertencentes ao caminho. Isto se deve ao fato de se estar considerando que os pesos associados aos nós terminais correspondam ao número de ocorrências do símbolo associado ao referido nó.

Como foi provado na seção 4, uma árvore que não é de Huffman pode ser fàcilmente detectada verificando se a sequência ascendente dos pesos na árvore foi quebrada em algum ponto. O algoritmo de atualização consiste , pois, em efetuar mudanças de nós e sub-árvores de forma a manter sempre a sequência correta. Desta forma a árvore é alterada à medida que os símbolos são codificados.

O algoritmo pode ser executado de uma forma direta (processo bottom-up) se for executado após cada procedimento de codificação. De outra maneira o processo pode tornar-se mais complexo.

5.2. O Algoritmo

O algoritmo atualiza uma árvore de Huffman especialmente construída. Devido às comparações e trocas necessárias na execução do algoritmo, foi preciso introduzir uma série de apontadores como mostra a Figura 3.

7. Intercambiar os valores de a e b ;

8. Fazer PESO (a) = PESO (a) + 1 ;

9. Fazer a = TBLINK (a) ;

10. Se a não aponta para a raiz da árvore, então vá para 2 ; caso contrário, fazer PESO (a) = PESO (a) + 1 e fim ;

Se um nó é a raiz de uma sub-árvore, a dupla troca no passo 6 envolve toda a sub-árvore. Veja Figura 4.

Prova : A árvore de Huffman original sòmente é alterada no passo 6. A decisão para que se execute ou não esta alteração é tomada no passo 3. Neste passo, dois casos podem ocorrer :

Caso 1. PESO (b) $>$ PESO (a) . Como o peso de um nó representa o número de ocorrências do referido nó, este valor deve ser inteiro. Portanto se PESO (b) $>$ PESO (a) , então PESO (b) $\geqslant$ PESO (a) + 1. Assim, após o passo 8, onde se faz PESO (a) = PESO (a) + 1, tem-se PESO (b) $\geqslant$ PESO (a). A ordem ascendente é, pois, mantida e não há necessidade de trocas.

Caso 2. PESO (b) = PESO (a). Deve-se procurar c tal que PESO (c) $>$ PESO (a) enquanto b é atualizado para satisfazer c = FLINK (b) . O intercâmbio que então se efetua entre nós ou sub-árvores apontadas por a e b não destroem nenhuma propriedade da árvore, pois PESO (a) = PESO (b). (Veja Lema 3.1.). Como PESO (c) $\geqslant$ PESO (a) + 1 recai-se no caso 1 e após o passo 8 a ordem ascendente é mantida.

Mantendo-se a ordem ascendente, mantem-se as propriedades da árvore de Huffman.

A troca entre um nó e seu pai pode destruir a árvore pois um nó terminal poderia tornar-se um nó interno. Para evitar este problema, impõe-se a restrição de se trabalhar apenas com pesos positivos.

A simulação mostrou que estes valores , em média, aproximam-se do melhor caso.

Os apontadores RLINK e LLINK são os comumente utilizados nas árvores binárias indicando respectivamente os nós filhos à direita e à esquerda.

O apontador TBLINK permite percorrer a árvore em ordem inversa , isto é, partindo-se de um nó terminal, atingir a raiz.

Os apontadores FLINK e BLINK indicam o nó cujo peso tem valor contíguo superior (FLINK) ou inferior (BLINK) na sequencia total de pesos dos nós.

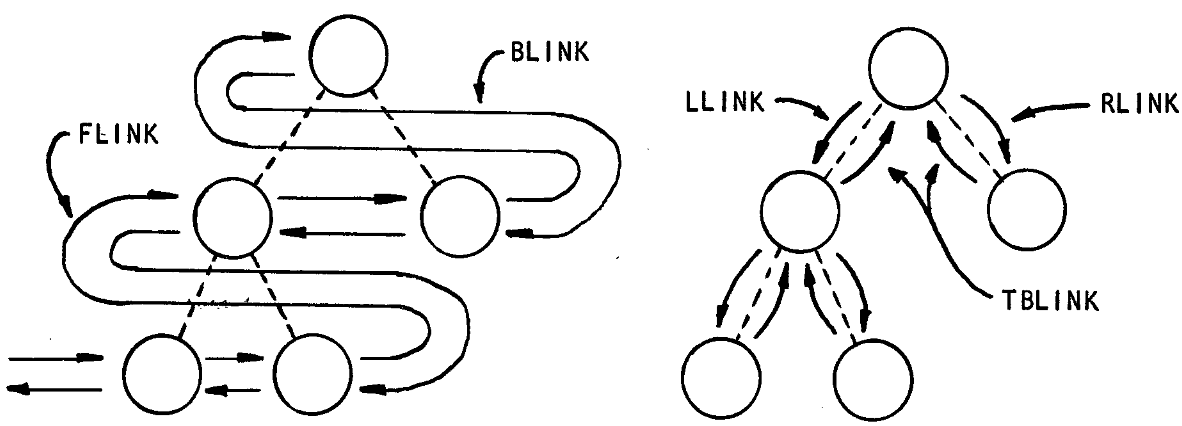


FIGURA 3. APONTADORES UTILIZADOS PELO ALGORITMO DE ATUALIZAÇÃO. MOSTRADO EM DOIS DESENHOS PARA FACILITAR A VISUALIZAÇÃO.

O algoritmo de atualização é efetuado partindo-se de um nó terminal e seguindo-se o caminho na árvore até atingir a raiz.

No algoritmo abaixo, a , b e c são apontadores.

1. Fazer a apontar para o nó terminal correspondente ao símbolo que se deseja codificar ;

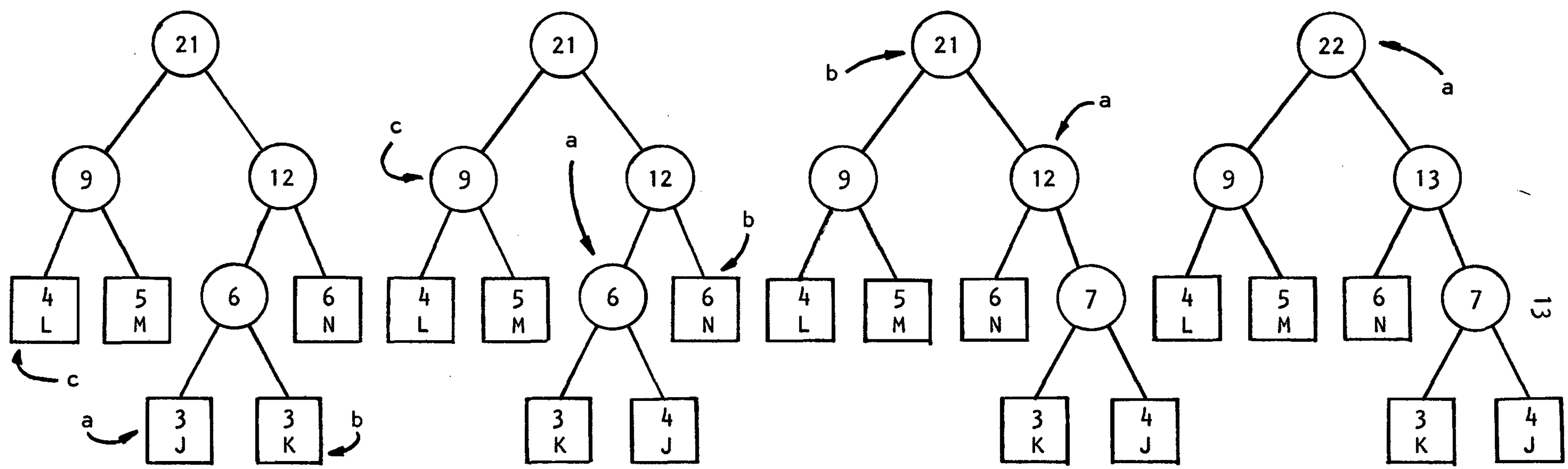
2. Fazer b = FLINK (a) ;

3. Se PESO (b) > PESO (a), então vá para 8 ;

4. Fazer c = FLINK (b) ;

5. Se PESO (c) = PESO (a), então fazer b = c e vá para 3 ;

6. Intercambiar os nós ou sub-árvores apontados por a e b ;



(a) ÁRVORE ORIGINAL (b) EM ADAPTAÇÃO (c) EM ADAPTAÇÃO (d) ÁRVORE ATUALIZADA

FIGURA 4. ÁRVORE DE HUFFMAN SENDO ATUALIZADA ENQUANTO O SÍMBOLO J É CODIFICADO.

5.3. Performance

Como estimativa de performance foram examinados o número de comparações (passos 3 e 5) e o número de trocas (passo 6). O estudo da performance do algoritmo não foi exaustivo.

À medida que o algoritmo adapta-se para minimizar o número de bits codificados, minimiza também o número de passos para a sua execução.

Desta forma, pareceu razoável examinar-se o comportamento do algoritmo nos dois casos extremos.

O pior caso parece ocorrer numa árvore completamente desbalanceada onde tem-se duas comparações a cada passo executado e a respectiva troca. Assim, tem-se : $2n - 2$ comparações e

$n - 1$ trocas

O melhor caso é alcançado depois de algum tempo se as características de geração dos símbolos da fonte não são alterados. Assim, tem-se :

custo normalizado da árvore

$$\frac{\sum w_i \, l_i}{\sum w_i} \leqslant \log_2 n$$ em comparações e nenhuma troca.

6. O SISTEMA

6.1. Introdução

Um modelo de um sistema para codificação e decodificação foi implementado usando-se como base o algoritmo da seção 5. O sistema pode ser visualizado simbòlicamente pela Figura 5.

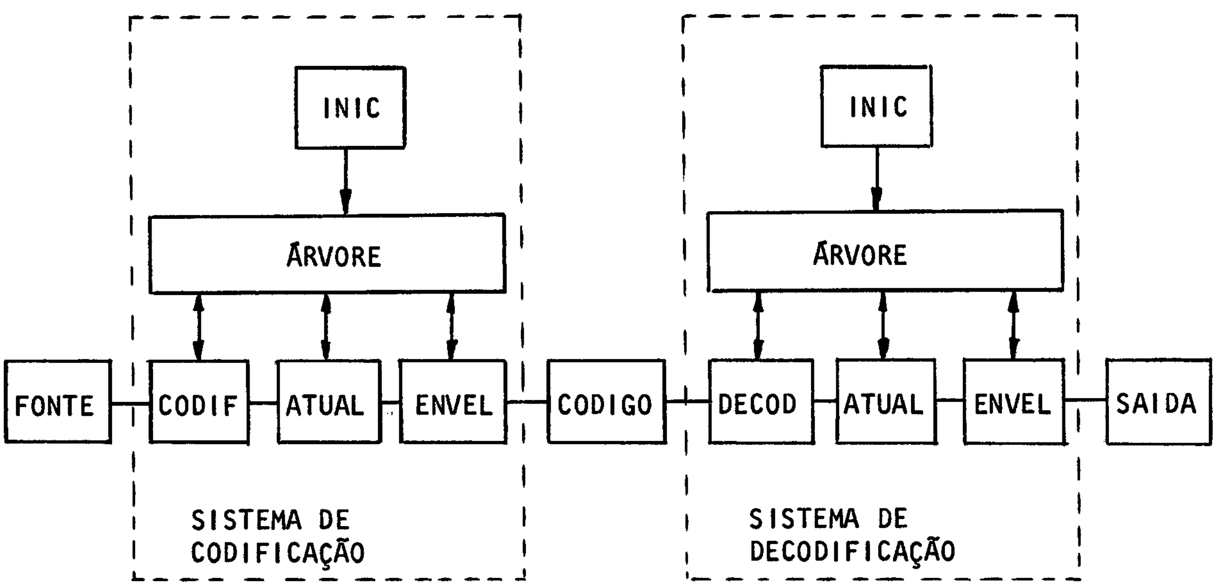


FIGURA 5. DIAGRAMA DE BLOCOS DO MODELO IMPLEMENTADO.

A FONTE gera uma sequência de símbolos que são codificados no <u>SISTEMA DE CODIFICAÇÃO</u> transformando-os em <u>CÓDIGO</u>. O <u>SISTEMA DE DECODIFICAÇÃO</u> por sua vez recebe a sequência de bits codificados e transforma-os na <u>SAÍDA</u> que contem a mesma informação gerada pela FONTE. Os sistemas de codificação e decodificação são compostos de uma série de algoritmos e tabelas, muitos dos quais são iguais nos dois sistemas.

Assim o bloco INIC cria e inicializa uma ÁRVORE de Huffman que será utilizada como uma tabela de símbolos para codificação e decodificação (Veja seção 6.2.). Os blocos CODIF e DECOD fazem respectivamente a codificação e decodificação dos símbolos utilizando a ÁRVORE como referência.

O bloco ATUAL utiliza o algoritmo apresentado em 5.2. e atualiza a árvore à medida que símbolos são codificados ou decodificados.

O bloco ENVEL executa um envelhecimento nos valores acumulados (Veja seção 6.3.).

Vale a pena lembrar que os blocos INIC, ÁRVORE, ATUAL e ENVEL são idênticos tanto na codificação como decodificação.

As características da FONTE utilizada estão na seção 7. O CÓDIGO pode ser armazenado num dispositivo de memória secundária ou transmitido através de um canal de comunicações.

6.2. A Árvore Inicial

No início do processo não se possui dados sobre as probabilidades do número de ocorrência dos símbolos a serem codificados. Sabe-se simplesmente qual o conjunto de símbolos possível. Heurísticamente parece ser razoável começar-se o processo com uma árvore de nós terminais com pesos iguais. Isto, teòricamente, significa que todos os símbolos são equiprováveis.

Lema 6.2.1. Numa árvore de Huffman de n elementos terminais de pesos iguais e não nulos, temos :

$2n - 2^m$ elemento no último nível

e $2^m - n$ elementos no penúltimo onde m é tal que $n \leq 2^m < 2n$.

Prova : Pela seção 4 sabemos que os pesos devem ficar em ordem ascendente nível a nível. Portanto, os elementos terminais devem distribuir-se no máximo em dois níveis pois o peso do maior elemento do penúltimo nível é o dôbro dos do último.

Temos dois casos :

1. $n = 2^m$ todos os nós terminais ficam num mesmo nível

2. $n \neq 2^m$ os nós terminais distribuem-se em dois níveis

Como em todos os níveis deve existir um número potência de dois nós, salvo no último, tem-se :

$$x + y = n$$

$$x/2 + y = 2^m$$

onde x = nós terminais no último nível
y = nós terminais no penúltimo nível.

Resolvendo o sistema $\quad y = 2^m - n$
$\quad x = 2n - 2^m$

Como o número de nós deve ser positivo ou nulo :

$$2^m - n \geqslant 0 \qquad 2n - 2^m > 0 \qquad n \leqslant 2^m < 2n$$

Sabendo-se o número de nós no último e penúltimo nível e que de nível para nível o número de nós cai a metade fica fácil estabelecer-se um algoritmo de construção direta da árvore de pesos iguais. (Veja Apendice 1 Rotina Montarv).

6.3. Processo de envelhecimento

6.3.1. Introdução

O algoritmo de atualização acumula o número de ocorrências de cada um dos símbolos codificados. Com base nestes números faz a atualização da árvore permitindo assim que ela se mantenha sempre com custo mínimo.

Neste processo adaptativo depara-se com duas características importantes :

1. Grau de otimização
2. Adaptabilidade

Como uma pode interferir na outra cumpre pois estabelecer um ponto de equilíbrio ótimo. O gráu de otimização máximo é obtido quando a frequência relativa de ocorrências de um determinado símbolo é igual a sua probabilidade. Isto tende a ocorrer depois de um grande número de símbolos codificados.

Por outro lado, diz-se que o sistema é mais adaptável quanto menor a diferença entre o número de ocorrências do símbolo mais referenciado e o do menos referenciado. Isto deve-se ao fato de se tornar o sistema mais sensível a quaisquer mudanças nas características da fonte e assim poder adaptar-se mais ràpidamente.

Como estas duas características são em parte conflitantes, cumpre pois, estabelecer um ponto de equilíbrio ótimo. Para isso deve-se permitir um acúmulo do número de ocorrências tal que permita uma otimização razoável sem que no caso de mudança na ocorrência dos símbolos torne o processo de adaptação muito lento. A idéia é simplesmente não se deixar a diferença do número de ocorrência dos símbolos mais frequentes e dos menos frequentes se tornar muito grande.

O processo utilizado é heurístico e nenhum desenvolvimento foi efetuado no sentido de se conseguir o melhor.

6.3.2. Algoritmo de envelhecimento

A filosofia do método é simplesmente dividir com um certo cuidado todos os pesos da árvore por um número μ.

1. Parte-se do nó de menor peso (a).
2. Se a é terminal então

   PESO (a) = PESO (a)/μ e vá para 4 ;

3. PESO (a) = PESO (RLINK(a)) + PESO (LLINK(a)), vá para 6 ;

4. Se PESO (a) $<$ 1 então PESO (a) = 1 ;

5. Se PESO (a) $<$ PESO (BLINK(a)) então PESO (a) = PESO (BLINK(a)) ;

6. a = FLINK (a) ;

7. Se a é raiz, então fim ;
   caso contrário vá para 2 ;

A árvore processada por este algoritmo continúa de Huffman e idêntica à original.

Prova : A estrutura da árvore não foi alterada , portanto é igual à original. A ordem ascendente é mantida pois :

Caso 1. Nós Internos :

$$\text{Supondo PESO (a)} \leqslant \text{PESO (b)} \leqslant \text{PESO (c)} \leqslant \text{PESO (d)}$$
$$\text{PESO (g)} = \text{PESO (a)} + \text{PESO (b)}$$
$$\text{PESO (h)} = \text{PESO (c)} + \text{PESO (d)}$$
$$\text{PESO (g)} \leqslant \text{PESO (h)}$$

Se

$$\text{PESO}'\text{(a)} = \text{PESO (a)}/\mu \quad \text{etc.,}$$

então

$$\text{PESO}'\text{(g)} = \frac{\text{PESO (a) + PESO (b)}}{\mu} \leqslant \frac{\text{PESO (c) + PESO (d)}}{\mu} = \text{PESO}'\text{(h)}$$

e a ordem ascendente é mantida.

Caso 2. Nós terminais :

O passo 5 garante a ordem ascendente.

Resta ainda discutir quando aplicar tal algoritmo. Isto, entretanto, foi feito arbitrariamente a cada v vezes em que o número médio de bits por símbolo aumentava. (Veja Seção 7.)

Isto parece ser razoável pois se o número médio de bits por símbolo está diminuindo, é bastante provável que o algoritmo esteja em fase de otimização e a aplicação de um processo de envelhecimento só viria retardar tal otimização. Por outro lado, se este número cresce é porque os símbolos que estão sendo referenciados tem um tamanho de código maior do que a média. Portanto, é provável que as características da fonte tenham se alterado. Um processo de envelhecimento vem, pois, acelerar a adaptação do algoritmo.

6.4. Implementação em computador.

O sistema foi implementado através de um programa em PL/I. Foi feito de forma modular para permitir alterações rápidas e fáceis no período de depuração e execução, além de facilitar muito a documentação do funcionamento de cada uma de suas partes.

A listagem se encontra no Apendice I.

A descrição das rotinas é a seguinte :

a. COMPRES

Programa principal. Simplesmente faz a chamada das diversas subrotinas em uma ordem determinada.

b. INICIO

Lê os diversos parâmetros que serão utilizados durante a execução do programa.

c. DEFARV

A partir do número de símbolos que serão utilizados para codificação, calcula os diversos parâmetros para a construção da árvore, além de alocar espaço de memória para a mesma.

d. MONTARV

Monta uma árvore com número de nós terminais especificados pressupondo pesos iguais.

e. ZERFREQ

Dada a estrutura da árvore, esta rotina associa peso um a cada nó terminal e calcula os pesos dos nós interno.

f. IMPARV

Imprime os valores contidos nos nós da árvore.

g. LERCA

Lê os símbolos a serem codificados. Isto é feito por leitura direta de um arquivo externo ou através da chamada de uma rotina que gere estes símbolos segundo uma lei determinada.

h. CODIF

Codifica o símbolo através de pesquisa na árvore.

i. ATUAL

Atualiza a árvore dependendo do símbolo referenciado, através do algoritmo da seção 5.

j. VELHO

Envelhece os pesos de cada um dos nós segundo o algoritmo da seção 6.

k. DECOD

Decodifica a sequência de bits através de pesquisa na árvore.

l) GRADIS

Grava a sequência de bits numa memória intermediária.

m) LERDIS

Lê a sequência de bits de uma memória intermediária.

n) FORMCOD

Formata a sequência de bits para permitir a gravação numa memória intermediária.

o) IMPCOD

Imprime superpostos o símbolo e a codificação correspondente.

p) RAND

Gera um número aleatório uniformemente distribuído entre 0 e 32767.

q) MARKOV

Gera, a partir de dois números aleatórios, um símbolo que será codificado. (Veja seção 7).

7. TESTES EFETUADOS

Dois tipos de teste foram efetuados para avaliar a performance do sistema.

No primeiro caso utilizou-se texto em português comumente empregado nos jornais. Os caracteres brancos não foram codificados.

No segundo caso foram utilizados dois conjuntos de 16 símbolos uniformemente distribuídos e gerados aleatoriamente. Os dois conjuntos estavam ligados pela matriz de probabilidades de transição :

$$M = \begin{pmatrix} x & 1-x \\ 1-x & x \end{pmatrix}$$

Os valores de <u>x</u> empregados foram 1,000 , 0,998 e 0.980. Desta forma, salvo para x = 1,000 , gera-se uma sequência de símbolos de comprimento aleatório pertencendo ora a um conjunto de 16 símbolos, ora a outro. Este tipo de teste foi escolhido porque o algoritmo aplica-se bastante bem ao caso em que os símbolos têm uma probabilidade de ocorrência bem determinada durante certo período de tempo.

Diversas constantes de envelhecimento <u>v</u> foram utilizadas (v = 8, 16 , 32 , 64 e ∞).

O fator μ , entretanto, sempre foi igual a <u>2</u>.(Seção 6.3.).

Os resultados em cada caso foram calculados após a codificação de aproximadamente 4000 símbolos e estão apresentados na TABELA 1.

O número médio de bits por símbolo que é o resultado mais importante , foi comparado com aquele obtido caso se tivesse conhecimento à priori da frequência relativa de cada símbolo a ser codificado. Desta forma poder-se-ia construir uma árvore de Huffman estática e através dela codificar o conjunto.

TIPO DE TESTE	x	v	valores por símbolo codificado					
			bits	comparações		trocas		
				máx.	méd.	máx.	méd.
TEXTO n= 48 AHE= 4,097	-	8	4,466	57	7,211	23	1,136
	-	16	4,331	55	6,179	23	1,016
	-	32	4,258	53	5,684	23	0,817
	-	64	4,223	53	5,193	23	0,573
	-	∞	4,165	53	4,726	23	0,258
GER. ALEAT. n= 32 AHE= 4,061	1,000	8	4,314	40	6,913	16	1,012
	1,000	16	4,160	40	5,574	11	0,722
	1,000	32	4,094	40	4,780	10	0,442
	1,000	64	4,087	40	4,742	10	0,424
	1,000	∞	4,100	40	4,693	10	0,389
GER. ALEAT. n= 32 AHE= 4,988	0,998	8	4,929	40	8,695	16	1,262
	0,998	16	4,615	40	7,633	11	1,121
	0,998	32	4,591	40	6,921	11	0,933
	0,998	64	4,687	40	6,610	10	0,886
	0,998	∞	5,017	40	5,768	10	0,510
GER. ALEAT. n=32 AHE= 5,000	0,980	8	5,048	37	9,115	9	1,320
	0,980	16	4,998	37	10,043	10	1,529
	0,980	32	5,017	37	8,343	8	1,285
	0,980	64	5,000	37	7,352	8	1,048
	0,980	∞	5,036	37	6,053	8	0,568

TABELA 1. RESULTADOS DA SIMULAÇÃO. (AHE= ÁRVORE DE HUFFMAN ESTÁTICA)

Vê-se pela TABELA 1 que para todos os casos o sistema produz resultados bastante próximos aos da árvore estática, lembrando-se, entretanto, que o sistema não necessita saber de qualquer propriedade acêrca da distribuição de probabilidades dos símbolos.

Para x = 0.998 o sistema produz menor número de bits que a árvore estática. Isto deve-se ao fato de haver tempo suficiente para adaptação momentânea do sistema a cada um dos conjuntos de 16 símbolos, enquanto que o cálculo da frequência relativa mascara estas características.

O número médio de comparações e trocas necessárias dá uma medida da quantidade de processamento necessária para a execução do algoritmo de atualização. Pode-se ver que o número máximo é bastante inferior ao máximo teórico no pior caso, enquanto que o número médio aproxima-se das previsões teóricas no melhor caso. (Seção 5.3.).

8. CONCLUSÕES

O sistema parece ser bastante atrativo para executar transmissão de dados entre computadores. No caso de $x = 0{,}998$ (Veja TABELA 1) pode-se codificar dados com um número de bits inferior ao conseguido com uma árvore de Huffman estática.

O sistema não precisa saber a priori qual o tipo de dados que serão codificados para minimizar o número de bits. Por esta razão êle é muito interessante para ser utilizado junto a fontes com alto grãu de imprevisibilidade.

Como sugestão para futuros desenvolvimentos pode-se citar um estudo detalhado sobre o processo de envelhecimento ou o comportamento do algoritmo sob condições especiais. A aplicação deste tipo de algoritmo à árvores ótimas construídas segundo restrições pode conduzir a resultados bem interessantes.

BIBLIOGRAFIA

(1) J. F. YOUNG, Information Theory, Butterworth, London 1971.

(2) D. A. HUFFMAN, A Method for the Construction of Minimum Redundancy Codes, Proc IRE, SET 52.

(3) D. E. KNUTH, The Art of Computing Programming, Vol. 1, Fundamental Algorithms, Addison Wesley 1968.

(4) T. C. HU & K. C. TAN, Path Length of Binary Search Trees, SIAM Applied Math., Vol 22 - nº 2, MAR 72.

APÊNDICE I

Devido aos inúmeros tipos de testes executados, tornou-se necessário fazer muitas modificações no programa original, a fim de se computar as estatísticas mostradas na Tabela 1.

Acrescido do fato de serem extremamente rigorosas as normas de apresentação dos originais de tese, não permitindo por exemplo cópias com redução de listagens de computador, tornou-se muito difícil a anexação da listagem do programa.

Mesmo não sendo imprescindível para o entendimento de todo o conteúdo da tese, as referidas listagens poderão ser conseguidas diretamente com o autor , através do NCE.