ANÁLISE E IMPLEMENTAÇÃO DE ALGORITMOS DE COMPRESSÃO DE DADOS

Maria Carolina de Souza Santos[1]
Orientador: Prof.º Ms. Mauricio Duarte[2]

Centro Universitário Euripides de Marilia – UNIVEM
FATEC – Faculdade de Tecnologia de Garça
Caixa Postal 17400-000 – Garça - SP – Brasil

mariacarolinasouzasantos@gmail.com
maur.duarte@gmail.com

Resumo: Em muitas situações são necessárias o armazenamento e o envio de informações pertinentes à alguma ação e, em muitos casos, por se tratarem de arquivos digitais muito grandes, estas informações necessitam ser compactadas (comprimidas). O ato de compactar uma informação é reduzir seu tamanho original em um tamanho menor, sem interferir no significado da informação. Este artigo apresenta uma técnica de compressão de dados proposta por Huffman e, baseando-se nela, propõe um novo algoritmo para compressão de dados.

Palavras-chave: 1. Compressão; 2. Huffman; 3. Algoritmos

INTRODUÇÃO

Compressão de texto está relacionada com as maneiras de representar o texto original em menos espaço. Para isto, basta substituir os símbolos do texto por outros que possam ser representados, usando um número menor de *bits* ou *bytes*. (ZIVIANI, 2007).

Assim, a compressão de dados é um processo de redução do espaço ocupado por dados num determinado dispositivo. O objetivo central é diminuir a quantidade de bytes do dado sem causar alterações, ou seja, utilizar compressão de dados sem perdas.

A pesquisa se constitui na proposta de criar um padrão de árvore de codificação, utilizando os conceitos abordados por Huffman, que é um dos métodos de compressão mais conhecidos, cujo objetivo é deixar no nível mais baixo da árvore as palavras de menor frequência, que, por consequência, geram códigos binários mais curtos para palavras mais decorrentes.

O estudo de compressão é importante para reduzir dados como imagens, vídeos e textos em situações como transmissão de dados, pois o ideal é diminuir o tempo de latência, e também para *backup*, que além de utilizar o conceito de compactação necessita fazer a compressão dos dados; caso os *backups* sejam feitos com frequência. Com a compressão, o arquivo compactado não ocupa tamanhos elevados nos dispositivos de armazenamento.

[1] Discentes do curso Bel. em Ciência da Computação UNIVEM - Marilia.
[2] Docente dos cursos de Análise e Desenvolvimento de Sistemas da FATEC – Garça e também docente do curso de Bel. em Ciência da Computação – UNIVEM - Marilia

1. REVISÃO BIBLIOGRÁFICA

1.1 COMPRESSÃO DE DADOS

Todos os dados computacionais como texto, músicas, imagens e vídeos são compostos por uma série de *bits*, sendo *bit* a menor unidade de armazenamento, que podem ser representados por *bytes*, que é o conjunto de oito *bits*.

Com o intuito de uma unidade de armazenamento ocupar o menor espaço possível, comprimir dados, ou seja, reduzir o tamanho das informações, esse procedimento tornou-se necessário. Atualmente, existem vários métodos de compressão, alguns deles utilizados de forma genérica, para vários tipos de dados, e outros de forma especifica.

Muitos desses métodos utilizam o conceito de substituição de informação, pois substituem uma quantidade de informação por uma de tamanho menor. Dessa forma, o dado comprimido ocupa menos espaço de armazenamento e, consequentemente, menos tempo para ser lido do disco ou ser transmitido por um meio de comunicação. Porém existem suas desvantagens, que até o momento são inevitáveis, sobretudo em relação ao processamento: é custoso codificar e decodificar. Como exemplo desses métodos, podemos citar o de Huffman.

1.2 MÉTODO DE HUFFMAN

David Albert Huffman, em 1952, desenvolveu um método de compressão sem perdas, resultado de uma pesquisa de doutorado, cujo nome do artigo é "A Method for the Construction of Minimum-Redundancy Codes".

Sua descoberta hoje é utilizada para comprimir dados *bit* a *bit* como textos e imagens. Para comprimir imagens é necessário representar seus *pixels* (*Picture Element*, menor elemento de imagem ao qual é possível atribuir-se uma cor) com um número binário. Nesse caso utiliza-se a quantização que estabelece o nível de frequência de uso de cada pixel e, por intermédio desse processo, é possível declarar um código para cada nível. Tendo-se esse código é possível utilizar o método de Huffman para fazer a compressão e descompressão.

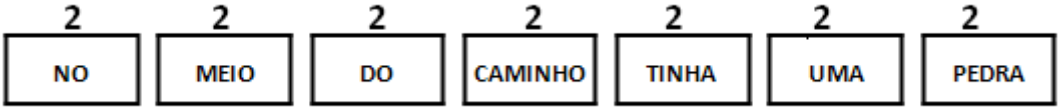
Para compressão de texto pode ser utilizado também o conceito de substituição de palavras por *bits*. Isso pode resultar em uma compressão maior na maioria dos casos. Se analisarmos o caractere "a" que em um texto pode representar uma palavra, o caractere por si só tem o tamanho de oito bits. Mas segundo o método de Huffman, que atribui códigos binários de tamanhos que variam de acordo com a frequência da palavra no texto, as de maiores frequências têm códigos binários menores e as de menores frequências têm códigos binários maiores: se "a" tiver uma freqüência menor, no pior caso pode ter tamanho maior que um byte, por outro lado, se a maior palavra da língua portuguesa como "pneumoultramicroscopicossilicovulcanoconiótico" (registrada no **Dicionário Houaiss** e reconhecida em 2001), que contém 46 caracteres, ou seja, 46 bits,

aparecer em um texto várias vezes existe grande probabilidade de reduzir seu tamanho e, no melhor caso, ser representada por um *bit*.

Huffman, em seu método, propôs produzir uma árvore de codificação que minimiza o tamanho do texto. As figuras ilustram um exemplo de construção dessa árvore, para isso a frase "No meio do caminho tinha uma pedra tinha uma pedra no meio do caminho", foi utilizada como exemplo.

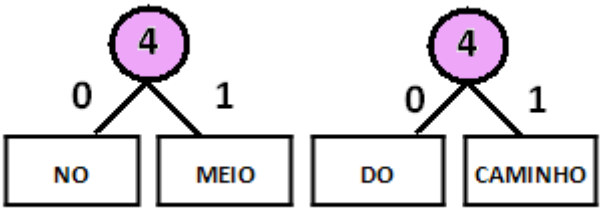
1º passo: As palavras são ordenadas em ordem crescente de frequência (Figura 1).

Figura 1 – 1º passo da construção da árvore.



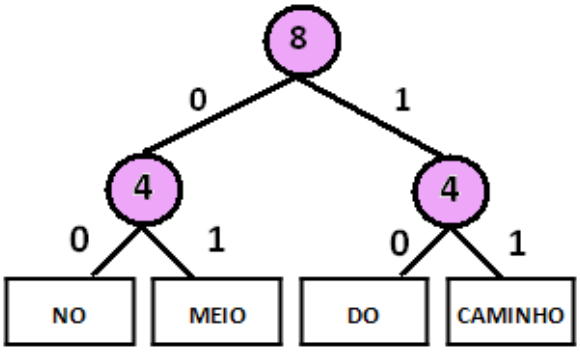
2º passo: As duas palavras de menor frequência devem formar uma nova sub-árvore (Figura 2).

Figura 2 - 2º passo da construção da árvore.



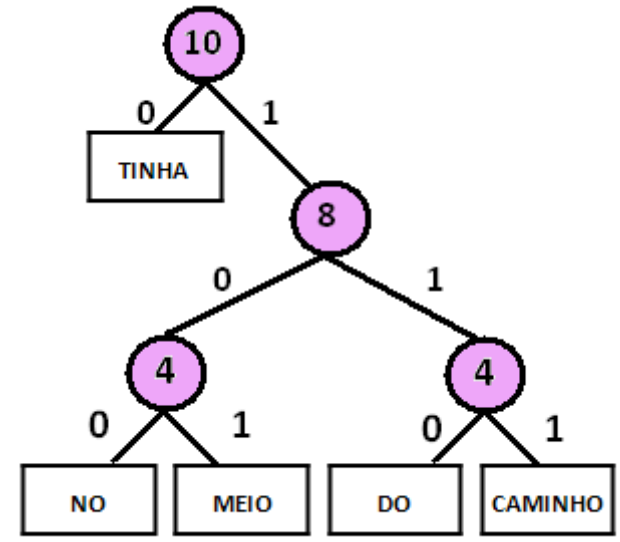
3º passo: Os nós são agrupados em pares (Figura 3).

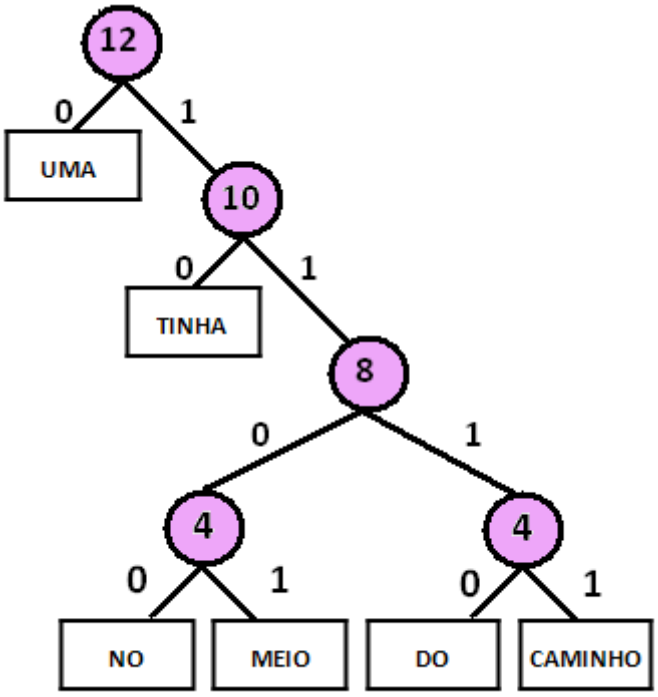
Figura 3 - 3º passo da construção da árvore.

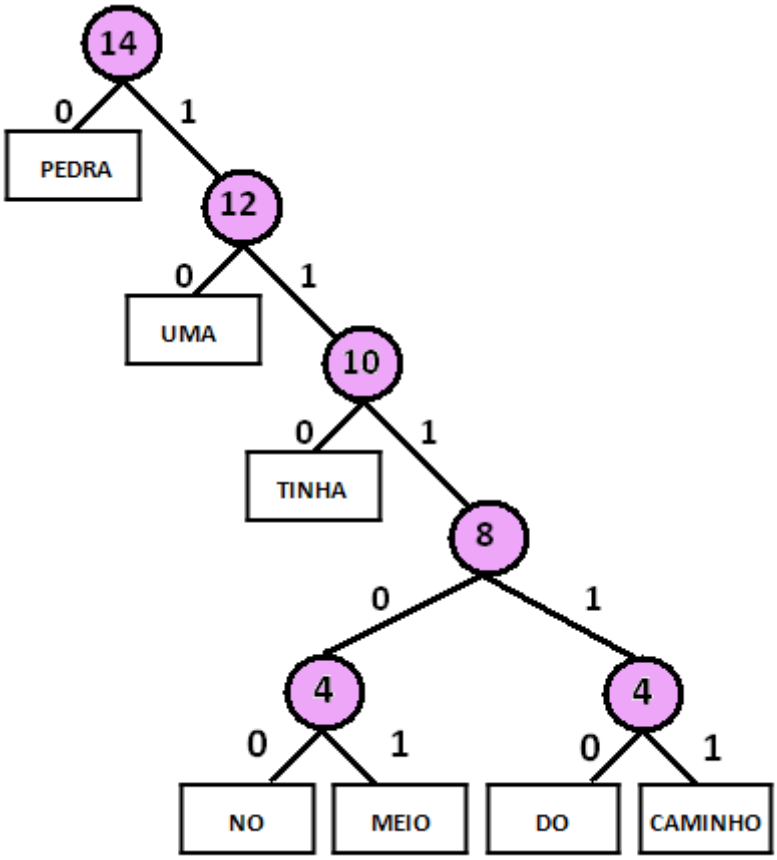


4º passo: Cada novo nó terá associado a ele uma nós palavra (Figura 4).

Figura 4 - 4º passo da construção da árvore.


10
0
1
TINHA
8
0
1
4
4
0
1
0
1
NO
MEIO
DO
CAMINHO


12
0
1
UMA
10
0
1
TINHA
8
0
1
4
4
0
1
0
1
NO
MEIO
DO
CAMINHO



Após a construção da árvore é possível determinar o código binário para cada palavra, para isso é necessário percorrer as arestas da árvore até a palavra que deseja obter a codificação e os valores atribuídos a cada aresta, sendo eles 0 e 1, formando o código binário. A tabela 1 mostra o resultado desse processo.

Tabela 1 – Palavras e seus respectivos códigos binários.

Palavra	***Código***
NO	11100
MEIO	11101
DO	11110
CAMINHO	11111
TINHA	110
UMA	10
PEDRA	0

O texto codificado é representando da seguinte forma:

11100 11101 11110 11111 110 10 0 110 10 0 11100 11101 11110 11111

A decodificação pode ser feita de duas formas: a primeira é utilizar a tabela que contém os códigos referentes a cada palavra e fazer a substituição; a segunda

forma é utilizar a árvore onde os *bits* de entrada do arquivo comprimido são usados para selecionar as arestas e, consequentemente, encontrar a palavra.

Tanto a primeira quanto a segunda forma geram um custo computacional expressivo, porém ainda há vantagens de comprimir o texto, se for feito a razão de compressão que é o tamanho do texto comprimido dividido pelo tamanho do texto original: 52 *bits* /552 *bits*, mostram que o arquivo comprimido foi reduzido para o tamanho de 9% do tamanho original.

2. METODOLOGIA

Para a implementação do algoritmo foi escolhida a linguagem Java.

Java é uma linguagem de programação orientada a objeto e desenvolvida na década de 90 por uma equipe de programadores, chefiada por James Gosling, na empresa *Sun Microsystems*.

Em 1999, com o surgimento das plataformas Java 2 *Enterprise Edition* (J2EE) e a Java 2 *Mobile Edition* (J2ME), Java passou a ser utilizada na web, em *desktop*, servidores, *mainframes*, jogos, aplicações móveis e até em *chips* de identificação, atualmente, por consequência de sua intensa aplicabilidade, é uma das linguagens mais usadas, servindo praticamente para qualquer tipo de aplicação.

Java executa em diferentes ambientes por trabalhar com uma máquina virtual nomeada *Java Virtual Machine* (JVM), que traduz o código *bytecode* em instruções específicas de cada sistema. Em relação ao *bytecode*, este é o estágio intermediário entre o código-fonte e a aplicação final, sendo ele transparente ao desenvolvedor. Portanto não se faz necessário entender como ocorre essa tradução para cada sistema, pois a máquina virtual é responsável por essa função, o que torna viável utilizar Java; afinal essa linguagem facilita o trabalho do desenvolvedor por não ter a necessidade de alterar o código da aplicação para diferentes ambientes.

A linguagem Java possibilita a criação de componentes separados por função, criando componentes mais simples e utiliza o conceito de herança, que são fatores importantes para o reuso do código, facilitando também o trabalho do desenvolvedor.

Outro motivo da escolha da linguagem é por ter ambientes de desenvolvimento gratuitos (IDE), tal como *NetBeans*, que é uma IDE para Java, e a mesma foi utilizada para implementação do algoritmo de compressão de dados. Além disso, vários fabricantes fornecem ferramentas, servidores de aplicação, bibliotecas e existem diversos tipos de *frameworks* para resolver um mesmo problema, o que proporciona maior liberdade de escolha de *framework* que mais se adapta ao projeto.

3. PROJETO

Projetar uma árvore com *n* palavras é vago, pois poderiam ter diversas representações. Pensando nisto é que este projeto propõe um novo padrão na construção da árvore de codificação de Huffman.

A construção da árvore é dividida em 4 etapas. Para exemplificar as etapas usaremos como exemplo o seguinte texto:

> *Depois de algum tempo você aprende a diferença , a diferença entre dar a mão acorrentar uma alma você aprende que amar não significa apoiar - se . que companhia nem sempre significa segurança ou proximidade .*

1ª Etapa:

Como foi proposto por Huffman, deve-se separar o texto por palavras e verificar a frequência das palavras.

Em seguida ordenar as palavras das de menores frequências para as de maiores, como na tabela 2.

Tabela 2 – Palavras e suas frequências em ordem crescente.

Ordem	Palavra	Freq.	Ordem	Palavra	Freq.
0	**Depois**	1	*16*	**!**	1
1	**de**	1	*17*	**companhia**	1
2	**algum**	1	*18*	**nem**	1
3	**tempo**	1	*19*	**sempre**	1
4	**,**	1	*20*	**segurança**	1
5	**entre**	1	*21*	**ou**	1
6	**dar**	1	*22*	**proximidade**	2
7	**mão**	1	*23*	**você**	2
8	**acorrentar**	1	*24*	**aprende**	2
9	**uma**	1	*25*	**diferença**	2
10	**alma**	1	*26*	**e**	2
11	**amar**	1	*27*	**que**	2
12	**não**	1	*28*	**significa**	2
13	**apoiar**	1	*29*	**.**	2
14	**-**	1	*30*	**a**	3
15	**se**	1			

2ª Etapa:

O nível mais baixo da árvore deve ser composto por pares.

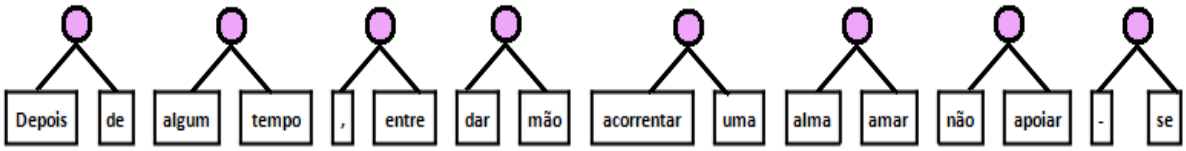
Para determinar a quantidade de palavras nesse nível, usa-se a seguinte equação:

$$p = \frac{n}{2}$$

Onde ***p*** é a quantidade de palavras no nível mais baixo e ***n*** é o número total de palavras. Se p for um valor impar acrescenta-se 1.Em seguida é necessário associar cada par a um nó.

O texto contém 31 palavras, portanto no nível mais baixo tem-se 15 palavras, como o valor é impar passa a ser 16 palavras e, consequentemente, 8 pares (Figura 5).

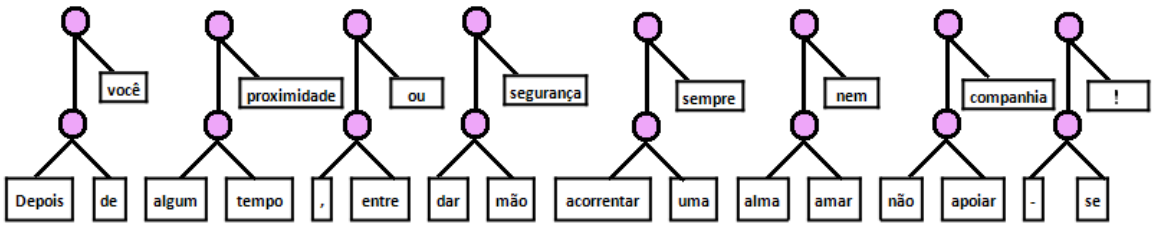
Figura 5 – Palavras no nível mais baixo agrupadas em pares.



3ª Etapa:

Nesta etapa estes pares gerados serão associados a um novo nó. O novo nó conterá o par na sua sub-árvore à esquerda e uma palavra em sua sub-árvore à direita. Esses novos nós deverão ser associados entre si de par em par também, criando outros nós (Figura 6).

Figura 6 – Nós e suas sub-árvores.



A 3ª etapa irá se repetir enquanto houver nós para serem associados em pares e palavras a serem inseridas na árvore.

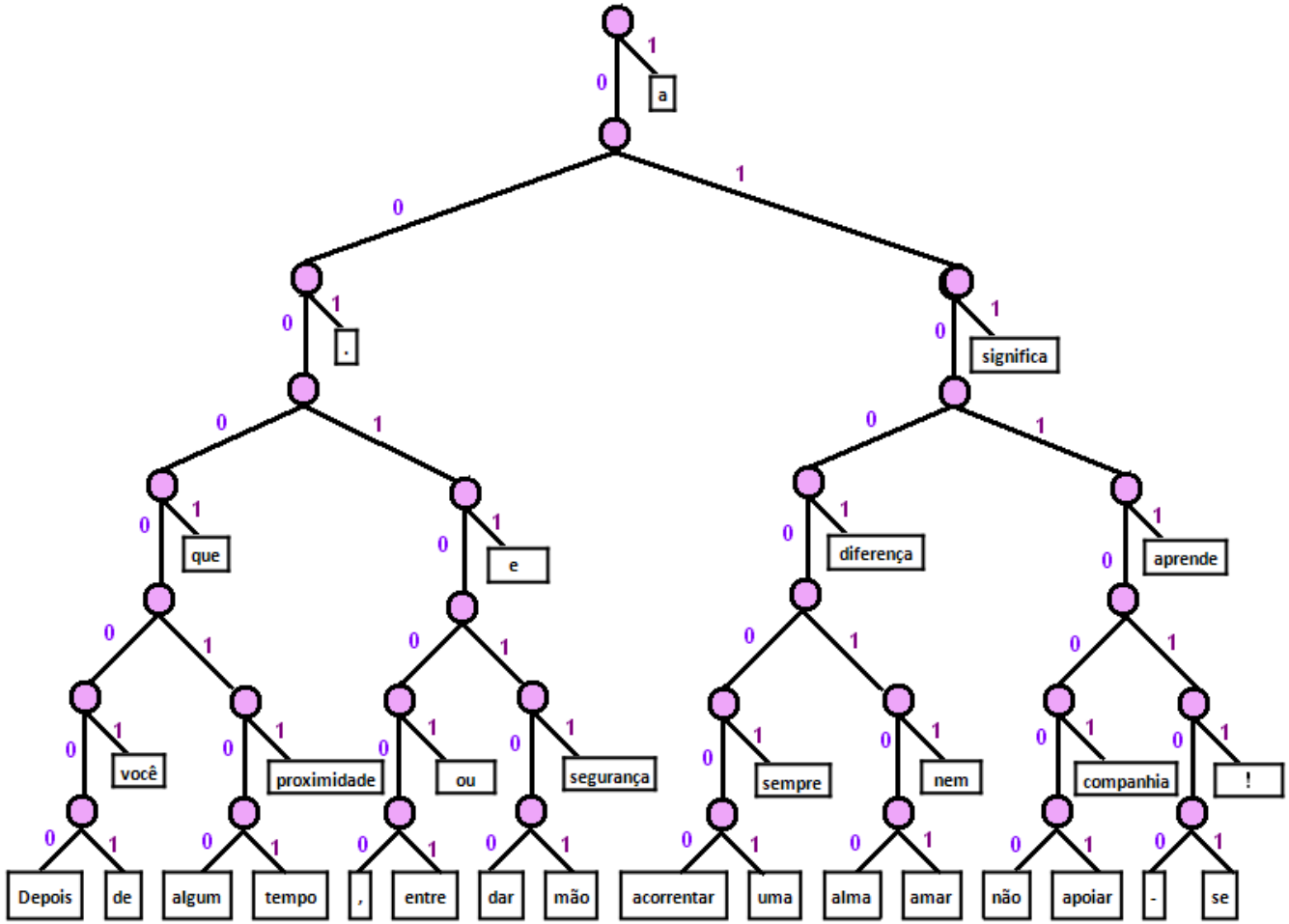
4ª Etapa:

Se a quantidade de palavras inseridas até o final da 3ª etapa for diferente do total de palavras, insere-se o restante acima da raiz.

$$r = n - ti$$

Onde ***ti*** é o total de palavras inseridas e ***r*** é a quantidade de palavras que serão inseridas acima da raiz (Figura 7).

Figura 7 – Árvore Final.



Após criar a árvore é possível determinar um código binário para cada palavra e mediante esse código é feita a descompressão.

Tabela 3 – Palavras e seus respectivos códigos binários.

Palavra	*Código*
Depois	00000000
De	00000001
Algum	00000100
Tempo	00000101
,	00010000
Entre	00010001
Dar	00010100
Mão	00010101
acorrentar	01000000
Uma	01000001
Alma	01000100
Amar	01000101
Não	01010000
Apoiar	01010001
-	01010100
Se	01010101

Palavra	*Código*
!	0101011
companhia	0101001
nem	0100011
sempre	0100001
segurança	0001011
ou	0001001
proximidade	0000011
você	0000001
aprende	01011
diferença	01001
e	00011
que	00001
significa	011
.	001
a	1

O texto codificado é representando da seguinte forma:

00000000 00000001 00000100 00000101 0000001 01011 1 01001 00010000 1 01001 00010001 00010100 1 00010101 01000000 01000001 01000100 0000001 01011 00001 01000101 01010000 011 01010001 01010100 01010101 001 00001 0101001 0100011 0100001 011 0001011 0001001 0000011 001

4. RESULTADOS E CONCLUSÕES

Analisando a tabela gerada pelo exemplo do projeto, um arquivo binário será gerado com as codificações descritas. Assim, é possível notar que as palavras menos frequentes foram codificadas em 7 bits e, no pior caso (um único caractere - possui 1 byte) economizou 1 bit. Em contrapartida, a maior palavra do texto “proximidade”, que possui 10 *bytes*, foi comprimida em 2 *bits*, indicando um grande redução do seu tamanho.

O arquivo original tem 209 *bytes*, no entanto, após aplicar o algoritmo proposto nesse projeto, ele ficou com 28,625 *bytes*. Ao fazer a razão de compressão: 28,625 *bytes* / 209 *bytes*, temos que o arquivo comprimido foi reduzido para o tamanho de 13% do tamanho original.

Para pequenos arquivos de texto é possível perceber que este padrão proposto tem uma taxa de compressão alta, porém para textos muito grandes essa taxa não é efetiva. Fica como proposta, para trabalho futuro, a análise e testes para arquivos maiores e o desenvolvimento de um padrão de árvore que se adapte à essa realidade.

REFERÊNCIAS BIBLIOGRÁFICAS

ZIVIANI, N. **Projeto de Algoritmos com implementações em Java e C++**. São Paulo: Cengage Learning, 2007.

TENENBAUM, A. M. et al. **Estruturas de Dados usando C**, Makron Books, 1995.