COMPAC8B: Compactação de arquivos através de sequências de 8 bytes.

Marcelo Nepomuceno da Silva[1], **Michelli Marlane da Silva**[1]

[1] Departamento de Ciência da Computação – Universidade Presidente Antônio Carlos (UNIPAC)
Rua Palma Bageto Viol S/N – Barbacena – MG – Brasil

mal2celo@gmail.com, michelli1983@yahoo.com.br

Abstract. *This article presents the main compactness techniques existing today. Through the literature review discusses the possibility of creating a new compacteness technique using statistical algorithms and characteristics of files to be compressed. With this analysis was developed a technique that differs from other, using standards 8 bytes reducing the total file size. Conclude that it is possible through comparision and tests performed on softwares existing that this method can be feasible for some types of files where the probability of sets repeated bytes becomes larger.*

Resumo. *O presente artigo se propõe a apresentar as principais técnicas de compactação existentes. Através da revisão bibliográfica discute a possibilidade da criação de uma nova técnica de compactação utilizando como base os algoritmos estatísticos e as particularidades dos arquivos a serem compactados. Por meio desta análise foi desenvolvida uma técnica que se difere das demais utilizando padrões de 8 bytes a fim de reduzir o tamanho total do arquivo. Concluímos que é possível através de comparações e testes realizados em softwares já existentes que este método pode ser viável para alguns tipos de arquivos onde a probabilidade de conjuntos de bytes repetidos se torna maior.*

1. Introdução

A compactação de dados é de fundamental importância para o futuro da computação. Segundo [Morimoto 2002] a compactação de arquivos sempre foi e ainda é um recurso muito utilizado, sua origem se confunde com a própria história da computação. Devido aos poucos e extremamente caros recursos de hardware disponíveis na época a compactação era fundamental. Através da compactação, era possível aumentar consideravelmente a quantidade de arquivos e programas que podem ser gravados no mesmo espaço físico. Alguns anos atrás, um HD de 200 Megabytes, por exemplo, poderia armazenar facilmente 300 a 400 Megabytes de arquivos compactados, um aumento considerável na capacidade de armazenamento. Conforme os discos rígidos e outras formas de armazenamento foram evoluindo e o seu custo diminuindo, o uso da compactação foi tornando-se menos crítico, mas este ainda é um recurso bastante utilizado hoje em dia.

Segundo [Hilbert and López 2011] a humanidade já possui mais de 295 hexabytes de informação. Considerando que 1 hexabyte é igual 1024 pentabyte e que 1 pentabyte equivale a 1024 terabytes, podemos observar que a humanidade já possui um número considerável de informações. E estes números tendem a crescer a cada ano.

Manter toda essa informação armazenada fisicamente gera altos custos, custos estes que podem ser tanto com a pesquisa, com o desenvolvimento ou com a manutencção destes equipamentos. Neste ponto podemos aplicar técnicas para compactar os dados antes de armazená-los. Ou seja, em um mesmo espaço de memória podemos armazenar uma quantidade a mais de informações. A compactação também pode ser utilizada na transmissão de dados auxiliando na diminuição do tráfego nas redes locais e até mesmo na rede mundial de computadores (Internet).

Percebendo que a compactação de dados é de fundamental importância para a computação, o presente trabalho pretende apresentar os métodos de compactação de arquivos já existentes e ainda propor um novo método de compactação, tendo por inspiração também o cálculo numérico com a representação numérica e conversões de bases. Assim, nosso objetivo final é o desenvolvimento de uma aplicação capaz de compactar e descompactar um arquivo, comparar os resultados com softwares já existentes, como WinRar, um software compactador e descompactador de dados desenvolvido em 1995 por Roshal, bem como o WinZip, que também compacta e descompacta arquivos sendo criado pela WinZip Computing em 1990, a fim de ilustrar a possibilidade da criação de novos métodos de compactação.

2. Estado da arte

Segundo [Morimoto 2002] compactar um arquivo é um simples processo de substituição. Por exemplo, cada caractere de texto ocupa 8 bits, onde um bit pode ser 0 ou 1 e em um conjunto de 8 bits é possivel representar 2^8 combinações distintas, o que nos dá um total de 256 combinações possíveis. O conjunto de caracteres ASCII prevê o uso de todas as 256 combinações, porém, em geral utilizamos apenas letras, números e acentuação. Já em uma imagem em .bmp com 256 cores, são usados também 8 bits para representar cada ponto, neste tipo de imagem pode existir grandes áreas com pontos da mesma cor. Por sua vez, em um arquivo executável sempre temos comandos e informações repetitivas. Em todos os casos se tem informações redundantes que poderiam ser perfeitamente substituídas por códigos menores.

Existem vários algoritmos que atuam diretamente nestas redundâncias, a fim de realizar uma compactação prevendo vários tipos de substituições para diferentes formatos de arquivos. Porém, uma vez compactado, um arquivo qualquer deixa de ser utilizável. Para poder usar novamente o arquivo, é preciso fazer o processo inverso para se obter o arquivo original. Ainda segundo [Morimoto 2002] existem três principais formas de compactação:

- Compactação de arquivos individuais baseada em um utilitário;
- Compactação de volumes;
- Compactação de arquivos feita pelo sistema operacional.

A Compactação de arquivos individuais baseada em um utilitário consiste em compactar arquivos utilizando programas como o WinZip, WinRar e etc. Utilizando estes aplicativos é possível perceber que alguns arquivos, como textos e certos tipos de imagens permitem uma taxa de compactação muito maior do que outros. Isto ocorre por que estes arquivos possuem muita informação redundante, por exemplo, uma imagem bitmap (.bmp) de 24 bits com o tamanho de 800 x 600 pixels com apenas um quadrado preto desenhado, terá após ser salva, um tamanho de aproximadamente 900 Kbytes. Após a

compactação a imagem ficará com apenas 3 ou 5 Kbytes, menos de 1% do tamanho original. O único problema é que, usando um destes programas compactadores de arquivos, será necessário descompactá-los antes de poder utilizá-los novamente. Estes programas estão sendo muito usados hoje em dia, principalmente na Internet, onde é comum compactar os arquivos antes de enviá-los, com o objetivo de poder agrupar todos os arquivos em um único diretório compactado a ser transmitido e principalmente objetivando diminuir o tempo da transferência.

Na Compactação de volumes ao invés de compactar arquivos individualmente, é possível criar volumes compactados, para tanto era possível usar programas como o DriveSpace (que acompanha o Windows 95/98). Em geral, é compactada uma partição de disco inteira. Todos os arquivos gravados nesta partição passam a fazer parte de um volume compactado, na verdade, um grande e único arquivo. Neste caso fica residente na memória um driver de compactação, que serve como um intérprete, compactando os arquivos antes de gravá-los e os descompactando conforme são lidos, entregando-os ao sistema operacional em sua forma original, tudo feito em tempo real.

Como os dados são gravados de forma compactada, em média é possível gravar 50 ou 60% a mais de dados. A desvantagem é que como o processador é utilizado para compactar/descompactar os arquivos, temos uma diminuição na performance geral do equipamento. O maior problema com este sistema, é que qualquer erro pode tornar o volume compactado inacessível, causando a perda dos dados gravados. Hoje em dia este sistema quase não é utilizado, também por que os programas disponíveis são capazes de trabalhar apenas em partições formatadas com FAT 16, não sendo compatíveis com FAT 32 e NTFS, por exemplo.

E por fim a compactação de arquivos feita pelo sistema operacional é um método permitido pelo Windows 2000 e Windows NT em partições NTFS, que permite unir melhor dos dois mundos, compactando individualmente arquivos ou pastas, mantendo os dados acessíveis, mas ao mesmo tempo economizando espaço em disco. Outra vantagem é que, devido aos arquivos serem compactados individualmente, não existe o risco de perda de dados.

Para que um dado possa ser compactado é necessário que o mesmo seja codificado. Segundo [Perna 1994] codificar um arquivo que está sendo representado por bytes consiste na modificação do sistema de representação por um outro sistema de representação a fim de que o mesmo arquivo possa ser representado por outro tipo de símbolo. Como por exemplo, uma sequência de caracteres. Ainda segundo ele, a codificação é base para muitas aplicações como:

- Reduzir o volume de dados do arquivo;
- Criptografá-lo;
- Auxílio na transmissão do arquivo em redes de computadores.

A compactação está presente em praticamente todos os aspectos computacionais. Como nos mostra [Santos et al. 2001], por exemplo em imagens médicas de alta resolução, imagens de grandes volumes que ocupam grandes quantidades de espaço de armazenamento, onde nenhuma perda de qualidade pode ser tolerável, devido à importância relevante dada aos detalhes diagnósticos, sendo assim, não podem ser comprimidas. Por outro lado técnicas de compactação e compressão com perda são convenientes para outras aplicações como em fotos comuns e vídeos.

A grande maioria dos métodos de compactação é baseada em duas metodologias [Perna 1994]:

- Método Baseado em Dicionário;
- Métodos Estatísticos.

Ainda segundo [Perna 1994] não é fácil definir o limite entre os métodos, pois existem métodos que utilizam mais de uma metodologia.

Os algoritmos que se baseiam em dicionários codificam sequências de símbolos, de comprimentos variáveis, representando-as por códigos. Esses códigos são utilizados na formação de índices. Índices estes, que são utilizados na descompactação dos dados. Métodos baseados em dicionários não necessitam conhecer as estatísticas dos dados a comprimir, podem utilizar sequências variáveis de símbolos.

Já os métodos estatísticos têm por objetivo diminuir o número de bits necessários para representar uma informação à medida que sua probabilidade de ocorrência aumenta [Perna 1994]. Ainda de acordo com [Arruda and Goes 2003] a fundamentação da compressão estatística é realizar uma representação otimizada de caracteres ou grupos de caracteres. Caracteres com maior probabilidade de ocorrência são representados por sequências binárias pequenas, e os com menor probabilidade de ocorrência são representados por sequências proporcionalmente maiores. Na compactação estatística, não é necessário saber qual caractere vai ser compactado, mas sim, ter o conhecimento da probabilidade de ocorrência de todos os caracteres sujeitos à compactação.

O presente trabalho se baseará nos dois métodos, tanto no método baseado em diciconário quanto no método estatístico. A medida que sequências de bytes se repetem são armazenadas em um arquivo auxiliar representadas por números inteiros, são armazenadas também as posições iniciais da ocorrência, bem como o número de repetições das mesmas. Sendo assim, a aplicação proposta realizará a criação de um dicionário, mas levará em conta os padrões de sequências de bits dos arquivos.

3. Materiais e métodos

O desenvolvimento da aplicação que utiliza a nova codificação valeu-se das seguintes ferramentas:

3.1. Linguagem de Programação Java

Foi utilizada a linguagem de programação Java. Segundo [Sebasta 2002] Java baseou-se em C++, mas foi projetado e desenvolvido para ser menor, mais simples e mais confiável.

Java cresceu a uma velocidade vertiginosa. Os programadores abraçaram a linguagem porque é mais simples do que sua rival mais próxima, C++. Além da própria linguagem de programação, Java tem uma rica biblioteca, o que torna possível escrever programas portáveis [Horstmann 2004].

3.2. Ambiente Integrado de Desenvolvimento

Apesar de ser possível o desenvolvimento Java através de editores de texto, com a compilação pelo prompt de comando, uma IDE de desenvolvimento pode ser de grande ajuda nesta tarefa. Segundo [Hubbard 2004] um ambiente integrado de desenvolvimento

(IDE) é uma coleção de programas colaborativos que facilitam o desenvolvimento de softwares. Na aplicação em questão foi utilizado o NetBeans IDE 6.9.1. O NetBeans é um programa Open Source para programadores escreverem, compilarem e depurarem programas escritos em Java, mas pode também fazer o mesmo com outras linguagens de programação como C, C++, PHP. A Sun Microsystem fundou o projeto Open Source NetBeans em junho de 2000 e continua sendo sua principal patrocinadora. O NetBeans IDE é um produto gratuito sem restrições de como ser utilizado.

4. Desenvolvimento

Para exemplificar como uma compactação de arquivos pode ser realizada, foi desenvolvida uma aplicação capaz de compactar e descompactar um arquivo. A tela principal da aplicação é apresentada na Figura 1 abaixo.

Figura 1. Tela principal da aplicação após a compactação de um arquivo.

A aplicação codifica o arquivo a ser compactado em conjuntos de oito bytes e faz uma busca por conjuntos repetidos, a medida que são encontradas repetições nos conjuntos de bytes uma referência ao conjunto é armazenada em um arquivo de configuração, juntamente com a sua posição e o número de repetições. Neste mesmo arquivo de configuração são armazenados mais alguns dados referentes ao arquivo original, para que a descompactação seja possível. Na primeira linha do arquivo de configuração é armazenada a extensão e na segunda linha é armazenado o tamanho do arquivo a ser compactado,

nesta mesma linha é armazenado caso seja necessário, uma referência aos bytes que não puderam ser codificados. O conjunto de bytes que não puderam ser compactados são montados em outro arquivo auxiliar. A descompactação se dá com a leitura do arquivo de configuração e a união dos bytes representados neste arquivo com os bytes restantes do arquivo auxiliar.

A codificação é iniciada com a leitura de todos os bytes do arquivo alvo da compactação, os mesmos são armazenados em um array de bytes. A aplicação verifica o tamanho deste array (tamanho do arquivo a ser compactado) e realiza uma divisão inteira deste valor por oito. O resultado da divisão será o tamanho do vetor onde será realizada a busca por repetições. O resto da divisão representa para a aplicação quantos bytes não poderão ”participar”da compactação, este número de bytes pode ser de zero a sete. Por exemplo, para um arquivo de 14.579 bytes serão armazenados os três primeiros bytes no arquivo de configuração pois o resto da divisão de 14.579 por 8 é 3. Para este mesmo exemplo teremos 1822 conjuntos de 8 bytes para serem comparados.

Para facilitar a tarefa de comparar os conjuntos de 8 bytes os mesmos sofrerão uma mudança de base, da base original 2 (binária) para a base 10 (decimal) podendo assim ser armazenados em um array do tipo long, que é um tipo primitivo do Java para armazenar números inteiros até 64 bits ou seja 8 bytes.

4.1. Compactação

A compactação acontece quando o vetor de longs entra em um laço de repetição, onde cada elemento é comparado com o seu sucessor, caso o elemento seja igual é marcada a posição do início da repetição e um contador armazena o número de repetições, seguindo assim até que apareça um elemento diferente ou termine de ler todos os elementos. Ao encontrar um elemento diferente, a posição, o número de repetições e o elemento são gravados no arquivo de configuração. Caso não haja repetição o elemento é inserido no arquivo auxiliar, onde vão ficar todos os longs ou conjuntos de 8 bytes que não foram gravados no arquivo de configuração. O laço segue até terminar de avaliar todos os elementos, ao final dois arquivos são gerados com as extensões .001 (arquivo auxiliar) para aqueles conjuntos de oito bytes aos quais a aplicação não encontrou repetições, e um segundo arquivo com extensão .002 (arquivo de configuração) que contém a extensão do arquivo original, o tamanho, os bytes de sobra da divisão inteira e a sequência de bytes repetidos.

A sequência de bytes repetidos é gravada da sequinte forma; Primeiro é gravada a posição onde se iniciou a repetição seguida de um espaço para demarcação, em seguida é gravado o número de repetições, com um espaço também para demarção e por fim é gravado o long que representa os oito bytes.

O pseudocódigo para o algoritmo de compactação é apresentado abaixo.

$busca \leftarrow Array[0]$
$cont \leftarrow 0$
$posicaoinicial \leftarrow 0$
for $i \leftarrow 1; i \leq tamArray; i \leftarrow i + 1$ **do**
  if $busca = Array[i]$ **then**
    $cont \leftarrow cont + 1$
  else

```
      if busca ≠ Array[i] and cont = 0 then
        InsereArquivo001(busca)
        busca ← Array[i]
        posicaoinicial ← posicaoinicial + 0
      else
        InsereArquivo002(posicaoinicial, cont, busca)
        busca ← Array[i]
        posicaoinicial ← posicaoinicial + 0
        cont ← 0
      end if
    end if
  end for
  if cont ≠ 0 then
    cont ← cont + 1
    InsereArquivo002(posicaoinicial, cont, busca)
  end if
```

4.2. Descompactação

A descompactação acontece com a leitura do arquivo de configuração, com extenção .002. Primeiro a extenção do arquivo a ser descompactado é lida e armazenada em uma variável temporária, depois o tamanho do arquivo é lido e armazenado em uma variável inteira, em seguida os bytes de sobra são lidos e armazenados em um vetor de bytes de no máximo 7 posições. Os conjuntos de oito bytes em forma de long são armazenados em uma matriz de três colunas e n linhas onde n é o número de linhas do arquivo menos dois, a primeira coluna recebe a posição de escrita dos bytes, a segunda coluna recebe o número de repetições da sequência e a terceira e última coluna recebe o conjunto de oito bytes no formato de um long. O arquivo auxiliar, com extenção .001, também é lido e armazenado em uma matriz de oito colunas e n linhas, onde n é o tamanho do arquivo dividido por oito.

As duas matrizes entram em um laço de repetição onde será montado o arquivo descompactado. Antes de entrar no laço, os bytes de sobra armazenados anteriormente são escritos neste arquivo. Em cada interação do laço é feito a busca pela posição de escrita na matriz de longs, caso encontre, o algoritmo converte o long para byte e entra em um laço com o número de repetições dos bytes, neste laço o conjunto de bytes é inserido no arquivo de destino e a posição de escrita é incrementada a cada interação, caso não encontre insere oito bytes da matriz que representa o arquivo de extensão .001 e incrementa a posição de escrita, estes passos são repetidos até o término das duas matrizes, neste momento o arquivo descompactado estará montado e pronto para ser novamente utilizado.

O pseudocódigo para o algoritmo de descompactação é apresentado abaixo.

```
extencao ← primeiraLinhaArquivo.002
tamanho ← primeiraPosicaoSegundaLinhaArquivo.002
ArraySobra[] ← demaisPosicoesSegundaLinhaArquivo.002
ArrayLongs[Tamanho_Arquivo.002][3] ← demaisLinhasArquivo.002
Array8Bytes[Tamanho_Arquivo.001][8] ← conjuntoBytesArquivo.001
posicaoEscrita ← 1
```

```
for i ← 0; i ≤ tamArraySobra; i ← i + 1 do
  MontaArquivo(ArraySobra[i])
end for
while do
  achou ← false
  posicao ← 0
  for i ← 0; i ≤ tamArrayLongs; i ← i + 1 do
    if posicaoEscrita = ArrayLongs[i][0] then
      achou ← true
      posicao ← i
    end if
  end for
  if posicaoEscrita < tamanho/8 then
    if achou = true then
      numeroRepeticoes ← ArrayLongs[posicao][1]
      posicaoEscrita ← posicaoEscrita + numeroRepeticoes
      for i ← 0; i ≤ numeroRepeticoes; i ← i + 1 do
        MontaArquivo(ArrayLongs[posicao][2])
      end for
    else
      MontaArquivo(Array8Bytes[cont])
      cont ← cont + 1
    end if
  else
    parar
  end if
end while
```

5. Resultados

A fim de ilustrar o funcionamento do método de compactação, esta seção trata dos testes realizados com o algoritmo de compactação. Todos os arquivos utilizados nos testes são de domínio público e podem ser encontrados em sites especialmente criados para este propósito, como o site http://www.dominiopublico.gov.br por exemplo.

O primeiro teste foi realizado com o arquivo .doc da Constituição Federal do Brasil de 1988. O arquivo original tem exatamente 1.071.616 bytes e após a compactação a soma do tamanho dos dois arquivos ficou em 1.041.584 bytes, uma redução de 30.032 bytes ou aproximadamente 2,8% do tamanho total do arquivo.

O segundo teste foi feito com o arquivo .doc do Hino Nacional Brasileiro. O arquivo tem exatamente o tamanho de 28.160 bytes e após a compactação a soma do tamanho dos dois arquivos ficou em 11.476 bytes. Chegando assim a uma redução de 16.648 bytes ou aproximadamente 59,25% do tamanho total do arquivo.

O terceiro teste foi realizado com o arquivo .bmp da Bandeira do Brasil. O arquivo tem exatamente o tamanho de 28.815 bytes e após a compactação a soma do tamanho dos dois arquivos ficou em 25.618 bytes. Chegando assim a uma redução de 3.197 bytes ou aproximadamente 11,10% do tamanho total do arquivo.

O quarto teste utilizou o arquivo .pdf do livro Dom Casmurro de Machado de Assis. O arquivo original tem exatamente 638.531 bytes e após a compactação a soma do tamanho dos dois arquivos ficou em 638.552 bytes. Resultando assim em um ganho de 21 bytes ou aproximadamente 0,003% do tamanho total do arquivo. Foi possível notar com este teste que para alguns tipos de arquivos o algoritmo não é capaz de compactar, podendo até mesmo acarretar em um ganho de bytes. Após a análise do arquivo de configuração (.002) podemos concluir que o ganho de 21 bytes se deu pela falta de padrões no arquivo original e foi a própria criação do arquivo de configuração que resultou no ganho e não na compactação.

O quinto teste foi feito com o arquivo .pdf do livro A Divina Comédia de Dante Alighieri. O arquivo original tem exatamente 1.797.156 bytes e após a compactação a soma do tamanho dos dois arquivos ficou em 1.797.078 bytes. Chegando assim a uma redução de 78 bytes ou aproximadamente 0,004% do tamanho total do arquivo. Podemos notar com este teste, que apesar do teste anterior feito com um arquivo .pdf ter resultado em um ganho de bytes, o mesmo não ocorreu para este arquivo, portanto não é a extenção do arquivo que influenciará no ganho ou na compactação, e sim as particularidades de cada arquivo a ser compactado.

O sexto teste foi realizado com um arquivo um pouco maior, o arquivo .mp3 do Hino Nacional Brasileiro. O arquivo original tem exatamente 2.833.833 bytes e após a compactação a soma do tamanho dos dois arquivos ficou em 2.779.641 bytes. Chegando assim a uma redução de 54.192 bytes ou aproximadamente 1,91% do tamanho total do arquivo.

O sétimo teste utilizou um arquivo .jpg, o 14 Bis de Alberto Santos Dumont. O arquivo original tem exatamente 29.365 bytes e após a compactação a soma do tamanho dos dois arquivos ficou em 29.355 bytes. Chegando assim a uma redução de apenas 10 bytes. Um resultado praticamente insignificante, mas quando comparado com os resultados dos outros softwares (WinRar e WinZip) se mostrou surpreendente. O WinRar aumentou o arquivo original em 71 bytes e seu resultado foi de 29.436 bytes, já o WinZip se mostrou um pouco menos eficiente e aumentou o tamanho do arquivo em 78 bytes, tendo como resultado final 29.443 bytes.

Já o oitavo e último teste foi realizado com um arquivo .jpg, o Balão de Santos Dumont. O arquivo original tem exatamente 33.809 bytes e após a compactação a soma do tamanho dos dois arquivos ficou em 33.798 bytes. Chegando assim a uma redução de apenas 11 bytes. Mais um resultado praticamente insignificante, mas quando comparado com os outros resultados, a compactação com o WinRar resultou em um arquivo com 33.900 bytes, um aumento de 91 bytes, já a compactação com o WinZip resultou em um arquivo com 33.931 bytes, um aumento de 122 bytes.

Podemos perceber que até mesmo aplicações comerciais dependem das particularidades (padrões nas sequências de bytes) de cada arquivo a ser compactado, os resultados do sétimo e oitavo teste confirmam esta afirmação.

A fim de comparações, todos os arquivos citados acima foram compactados com duas ferramentas bem populares, o WinRar que pode ser baixado em http://www.winrar.com e o WinZip que já vem nas distribuições do Windows. Os resultados são apresentados na tabela 1 abaixo

Tabela 1. Comparativo dos testes realizados.

Teste Nº	Nome do Arquivo	Tamanho (em bytes)			
		Original	Algoritmo	WinZip	WinRar
1	Constituição.doc	1.071.616	1.041.584	23.746	**23.679**
2	Hino brasileiro.doc	28.160	11.476	6.389	**6.230**
3	Bandeira do Brasil.bmp	28.815	25.618	23.746	**23.679**
4	Dom Casmurro.pdf	638.531	638.552	601.595	**599.499**
5	A Divina Comédia.pdf	1.797.156	1.797.078	1.528.214	**1.510.984**
6	Hino Nacional.mp3	2.833.833	2.779.641	2.735.070	**2.727.280**
7	14 Bis.jpg	29.365	**29.355**	29.443	29.436
8	Balão.jpg	33.809	**33.798**	33.931	33.900

Já as tabelas 2 e 3 apresentam as diferenças de tamanho final e porcentagem entre o algoritmo e o WinZip e entre o algoritmo e o WinRar

Tabela 2. Diferenças entre o algoritmo e o WinZip.

Teste Nº	Nome do Arquivo	Tamanho (em bytes)			
		Algoritmo	WinZip	Diferença	Porcentagem
1	Constituição.doc	1.041.584	23.746	1.017.838	97,72%
2	Hino brasileiro.doc	11.476	6.389	5.087	44,33%
3	Bandeira do Brasil.bmp	25.618	23.746	1.872	7,31%
4	Dom Casmurro.pdf	638.552	601.595	36.957	5,79%
5	A Divina Comédia.pdf	1.797.078	1.528.214	268.864	14,96%
6	Hino Nacional.mp3	2.779.641	2.735.070	44571	1,60%
7	14 Bis.jpg	29.355	29.443	-88	-0,30%
8	Balão.jpg	33.798	33.931	-133	-0,39%

Tabela 3. Diferenças entre o algoritmo e o WinRar.

Teste Nº	Nome do Arquivo	Tamanho (em bytes)			
		Algoritmo	WinRar	Diferença	Porcentagem
1	Constituição.doc	1.041.584	23.679	1.017.905	97,73%
2	Hino brasileiro.doc	11.476	6.230	5.246	45,71%
3	Bandeira do Brasil.bmp	25.618	23.679	1.939	7,57%
4	Dom Casmurro.pdf	638.552	599.499	39.053	6,12%
5	A Divina Comédia.pdf	1.797.078	1.510.984	286.094	15,92%
6	Hino Nacional.mp3	2.779.641	2.727.280	52.361	1,88%
7	14 Bis.jpg	29.355	29.436	-81	-0,28%
8	Balão.jpg	33.798	33.900	-102	-0,30%

Podemos observar após a análise da tabela 2, que apesar do algoritmo apresentado neste trabalho ser simples, o resultado da comparação para certos tipos de arquivos como .bmp e principalmente .mp3 se mostrou bem eficiente, atingindo uma diferença de aproximadamente 7% para o arquivo .bmp e de aproximadamente 2% para o arquivo .mp3

Ainda na tabela 2 podemos observar que até mesmo os softwares comerciais podem fugir ao seu propósito e aumentar um arquivo ao invés de compactá-lo. Este fato pode ser observado para os arquivos 14 Bis.jpg e Balão.jpg onde o algoritmo conseguiu ser 0,30% melhor para o primeiro e 0,39% melhor para o segundo, respectivamente.

Observando a tabela 3, é possível notar que o resultado se mantém praticamente inalterado quando é feita a comparação entre o algoritmo e o WinRar. chegando a uma diferença de aproximadamente 8% para o arquivo Bandeira do Brasil.bmp e de aproximadamente 2% para o arquivo Hino Nacional.mp3

O mesmo acontece para os arquivos onde o WinZip se mostrou ineficiente. O WinRar também aumentou os arquivos ao invés de compactá-los. E, novamente o algoritmo proposto neste trabalho se mostrou mais eficiente, conseguindo ser 0,28% melhor para o arquivo 14 Bis.jpg e 0,30% melhor para o arquivo Balão.jpg

Os quatro melhores resultados do algoritmo proposto neste trabalho de conclusão de curso são representados nas figuras 2, 3, 4 e 5 que demonstram de forma gráfica os resultados obtidos;

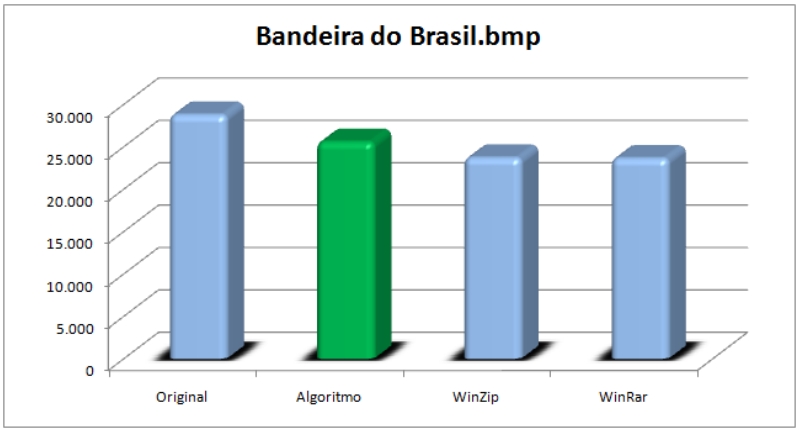

Figura 2. Comparativo dos resultados para o arquivo Bandeira do Brasil.bmp

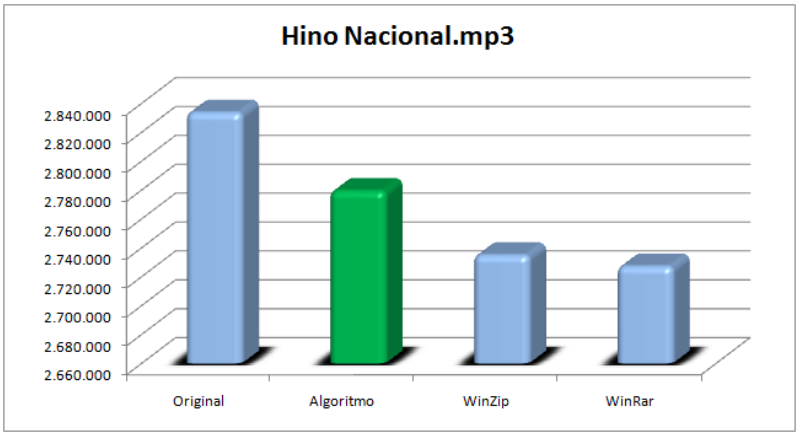

Figura 3. Comparativo dos resultados para o arquivo Hino Nacional.mp3

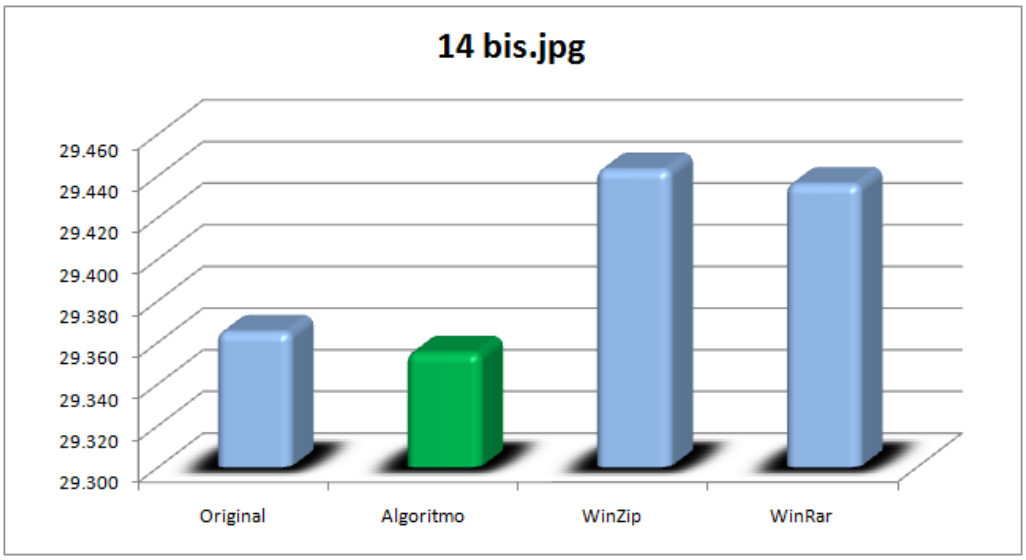

Figura 4. Comparativo dos resultados para o arquivo 14 Bis.jpg

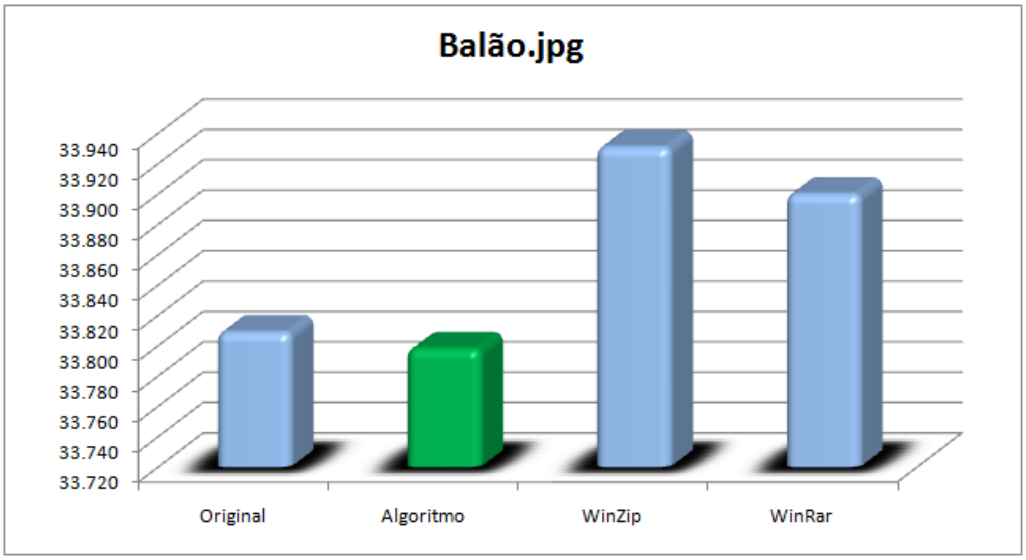

Figura 5. Comparativo dos resultados para o arquivo Balão.jpg

6. Conclusão

Foi possível concluir com base nos testes realizados, nos embasando nas análises dos resultados, e nas principais características dos métodos de compactação - métodos baseados em dicionários e métodos estatísticos - métodos estes que formaram a base para a aplicação apresentada neste trabalho, que tanto os softwares comerciais quanto a ferramenta implementada cometem falhas ao aumentar o tamanho final do arquivo alvo da compactação, pois dependem das características fundamentais do arquivo, fugindo assim ao seu propósito, inflando os arquivos. A diferença é que o algoritmo desenvolvido se mostrou mais eficiente para determinados tipos de arquivos. E que apesar de existirem vários métodos de compactação, cada um dos quais com suas particularidades, vantagens e desvantagens, foi possível observar que existe a possibilidade de se criar métodos mais

eficientes, como o algoritmo proposto neste trabalho, que obteve em alguns casos um resultado melhor do que as ferramentas comerciais testadas. Em nenhum momento nos propomos a mostrar que uma ferramenta é melhor ou pior que a outra, e nem de tentar desenvolver uma ferramenta mais eficiente do que as já existentes. O foco consistiu em mostrar a importância da compactação para o futuro da computação, já que nos dias atuais, a humanidade tende a armazenar cada vez mais um número maior de informações. Com este trabalho foi provado que é possível criar e aprimorar métodos de compactação, para isso os estudos devem ser continuados a fim de se melhorar o algoritmo apresentado, ou com base nele criar novos algoritmos. A melhor gerência no armazenamento das informações, pode trazer grandes benefícios para todos aqueles que a utilizam, esta gerência foi e sempre será peça chave para o avanço da tecnologia da informação, onde a ciência da computação atua decisivamente.

Referências

Arruda, D. V. O. and Goes, R. S. (2003). Programa para compactação de dados utilizando código de huffman. Master's thesis, Universidade Federal de Goiás, Escola de Engenharia Elétrica e de Computação.

Hilbert, M. and López, P. (2011). The world technological capacity to store, communicate, and compute information. *Science Xpress*.

Horstmann, C. (2004). *Big Java*. Bookman, 1 edition.

Hubbard, J. R. (2004). *Programação Com Java*. Bookman, 2 edition.

Morimoto, C. E. (2002). Hardware manual completo.

Perna, M. A. L. (1994). Módulo de compactação de imagens discretas. Master's thesis, Instituto Militar de Engenharia do Rio de Janeiro, Programa de Pós-graduação em Informática.

Santos, E. T., Zuffo, M. K., Netto, M. L., and de Deus Lopes, R. (2001). Computação gráfica: Estado da arte e a pesquisa na usp. In *15º Simpósio Nacional de Geometria Descritiva e Desenho Técnico*.

Sebasta, R. W. (2002). *Conceitos de Linguagem de Programação*. Bookman, 5 edition.