
**UNIVERSIDADE FEDERAL DE SANTA MARIA**

CURSO DE SISTEMAS DE INFORMAÇÃO

Disciplina de Pesquisa e Ordenação de Dados -
Departamento de Eletrônica e Computação (DELC)

---

# TRABALHO 2 - RELATÓRIO REFERENTE AO ARTIGO:

***An Improved Data Compression Method for General Data de Salauddin Mahmud***

---

*Alunos:*
Adriano Almeida
Emanoel de Moura
Leonardo Trindade
Marinara Rübenich

02 de Julho, 2018

# 1 Resumo

*Este relatório apresentará um resumo do artigo "**An Improved Data Compression Method for General Data**" escrito por Salauddin Mahmud e também o código desenvolvido pelos autores utilizando a linguagem C baseada nos experimentos deste mesmo artigo.*
*O capítulo 2 apresentará uma síntese do artigo e o capítulo 3 irá mostrar o desenvolvimento do programa e parte dos códigos que foram feitos*

# 2 Introdução

A compactação de dados vem sendo estudada há muitos anos, sendo muito útil principalmente nas tecnologias de comunicações, onde quanto menor os dados trafegados forem, melhor será a eficiência, pois, ela irá reduzir o espaço ocupado por quaisquer tipos de dados onde quer que eles se encontrem. Ou seja, através da compressão é possível reduzir os bits (ou bytes) com os quais representamos os dados.

O artigo "An Improved Data Compression Method for General Data", [Mahmud, 2012], propõe uma metodologia de compactação de dados gerais que se baseia em uma tabela verdade lógica. O autor afirma que cada dois bits de dados podem ser representados por um único bit. Antes de ilustrar esta afirmação, no próximo parágrafo vamos resumir a seção 2 do artigo.

Na seção 2, o autor diz que existem dois tipos de compressão: a compressão sem perdas e a compressão com perdas.

1. **Compressão sem perdas:** no momento da descompactação do dado será completamente mantido, ou seja, nenhum bit do arquivo original será perdido. Em geral, dados redundantes e consecutivos são substituídos por códigos que devem ser reconhecidos pelo descompactador. Existem diversos descompactadores disponíveis e os melhores e mais atuais utilizam modelos probabilísticos, previsões estatísticas e codificações aritméticas. É muito utilizada em planilhas, textos ou programas executáveis.

2. **Compressão com perdas:** a redução do arquivo elimina permanentemente alguns bits dos dados, porém, após a descompactação, isto se torna praticamente ou totalmente imperceptível ao usuário. Muitas técnicas são utilizadas para que, mesmo com as perdas, a descompressão não tenha um resultado que cause algum impacto notável. Como exemplo eliminar os sons menos audíveis pelos seres humanos em músicas ou remover uma cor muito semelhante a outra em vez da iluminação em uma fotografia.

## 2.1 Método de Compressão proposto por Mahmud

Ele propõe que os bits dos dados podem ser representados pela metade dos seus bits, ou seja, arquivos de 64 bits podem ser representados por 32 bits, de 32 bits podem ser representados por 16 bits, e assim sucessivamente, basta dividir o número de bits do arquivo original por 2 (64bits / 2 = 32 bits). O que também significa que 1GB de dados podem ser representados por sua metade: 512MB de dados.

Sua técnica baseia-se em uma tabela verdade lógica que junta 2 bits e os transforma em 1 bit só. Ela pode ser vista a seguir:

Tabela 1: Tabela Verdade proposta

| A | B | Z |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | $\bar{0}$ |
| 1 | 0 | $\bar{1}$ |
| 1 | 1 | 1 |

Fonte: [Mahmud, 2012]

Explicando melhor a tabela 1, quando **Z** for:

(a) **0:** significa que os dados A e B são 0 (00 = 0).

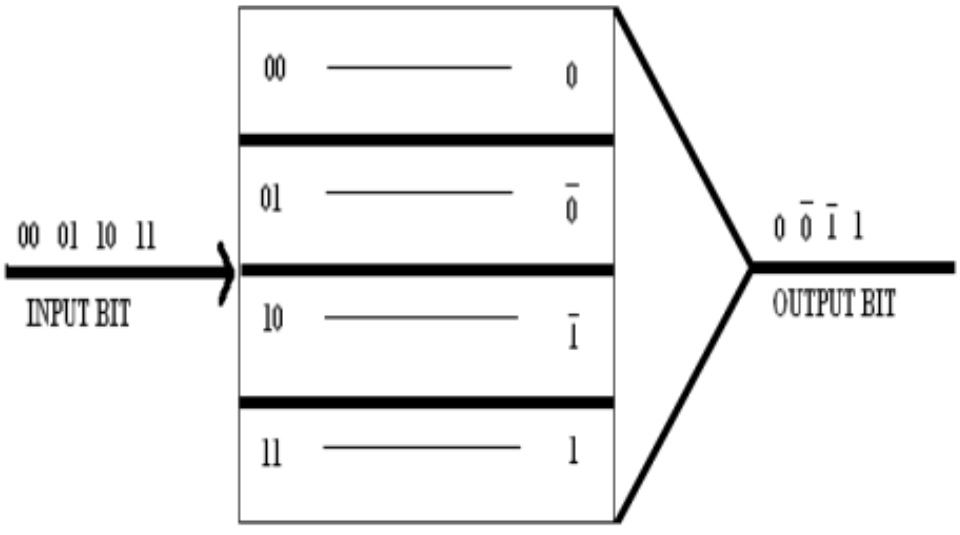
(b) $\bar{\mathbf{0}}$**:** significa que A é 0 e B é 1 (01 = $\bar{0}$).

(c) $\bar{\mathbf{1}}$**:** significa que A é 1 e B é 0 (10 = $\bar{1}$).

(d) **1:** significa que A é 1 e B é 1 (11 = 1).

A sequência de dados de entrada deve ser par para o algoritmo poder dar certo, se uma quantidade de dados ímpar é recebida, é necessário adicionar 1 bit ao final da sequência. Se o último valor for 0 um novo 0 é adicionado, senão adiciona-se o valor 1. A figura 1 ilustra como o método deve funcionar.

Figura 1: Representação da técnica proposta




Fonte: [Mahmud, 2012]

Então a apliação do método deve funcionar considerando o texto:

(a) Um valor X que recebe o array: X = {1,0,1,1,0,1,0,1,0,0,1,1,0,0,1,1,0,0,1,0}, onde X = 20.

(b) Um array com 20 valores onde cada posição recebe i, inicia em 1 e i é incrementado de 1 a 1: arr=array(0,1,0,1,0,0,1,1,0,1,1,0,1,0,1,0,1,0,1,0).

(c) Um valor Y que é gerado a partir do array arr randomizado (y =arr[rand]): Y = {01010011011010101010}, onde Y = 20.

(d) E finalmente o resultado Z que é proveniente do XOR (ou exclusivo) entre X e Y: Z = {1,1,1,0,0,1,1,0,0,1,0,1,1,0,0,1,1,0,0,0}, onde Z = 20. Aqui o arquivo já está criptografado e pronto para ser compactado.

O resultado da compactação é:

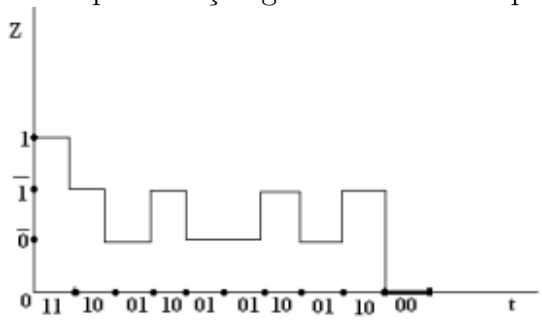
Tabela 2: Tabela mostrando a compactação

| 11 | 10 | 01 | 10 | 01 | 01 | 10 | 01 | 10 | 00 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | $\overline{1}$ | $\overline{0}$ | $\overline{1}$ | $\overline{0}$ | $\overline{0}$ | $\overline{1}$ | $\overline{0}$ | $\overline{1}$ | 0 |

Fonte: [Mahmud, 2012]

A tab Z representado em 10 bits agora. Temos também a demonstração em gráfico:

Figura 2: Representação gráfica da técnica proposta




Fonte: [Mahmud, 2012]

Para o cálculo de eficiência, o seguinte cálculo de taxa de compressão deve ser realizado:

$$Taxa = \frac{DadosCompactados}{DadosNãoCompactado} * 100\% \gg\gg Taxa = \frac{10}{20} * 100\% \quad (1)$$

O que resulta em uma taxa de compressão de 50%.
Também existe o fator de compressão conforme mostrado abaixo:

$$Compressão = \frac{TamBitsEntrada}{TamBitsSaída} \gg\gg Compressão = \frac{20}{10} = 2(2)$$

Fatores acima de 1 significam que houve redução nos dados.

# 3 Desenvolvimento

Como desenvolvimento foi desenvolvido um código na linguagem C que é capaz de fazer a compactação conforme a proposta de [Mahmud, 2012]. O mesmo será apresentado e explicado a seguir:

```
//PRINCIPAL
int main(){
    int tamVetor, gerar;
    int array[tamVetor];
    int x[20] = {1,0,1,1,0,1,0,1,0,0,1,1,0,0,1,1,0,0,1,0};
    int y[tamVetor]; //y gerado apartir do array
    int z[tamVetor]; // vetor criptografado
    int compacta[tamVetor/2]; // vetor comprimido
    int descompacta[tamVetor]; // vetor descomprimido
```

Aqui o vetor X se encontra conforme o proposto e as outras variáveis importantes, ARR, Y, Z, compactar e descompactar, são iniciadas.

```
//NUMEROS RANDOMICOS
int serieRandomica(int min, int max){
    return(rand() % (max - min)) + min;
}

// GERA ARRAY ALEATORIO
// O vetor aleatorio e gerado a partir de um random de 0 ou 1
int arrayAleatorio(int *array, int tamVetor){
     int i;

     for(i = 0; i < tamVetor; i++){
         int random = serieRandomica(0,2);
         array[i] = random;
     }
     return *array;
}

//GERA O "Y" A PARTIR DO ARRAY
int geraYRandom(int *y, int tamVetor, int *array){
    int i;

    for(i = 1; i < tamVetor; i++){
        int random = serieRandomica(0, tamVetor);
        y[i] = array[random];
    }
    return *y;
}
```

Nesta parte ocorre a geração do vetor Y, que é gerado a partir do ARR aleatório, tendo valores de 0 e 1, até chegar ao limite do tamanho máximo do array.

```
// GERA O VETOR Z
// Vetor Z e resultado do (Xor X and Y)
int vetorZ(int *z, int tamVetor, int *x, int *y){
    int i;
```

```
5
6       for(i = 0; i < tamVetor; i++){
7           int or = x[i]|x[i];
8           int and = (or != y[i]) ? 1 : 0;
9           z[i] = and;
10      }
11
12      printf("\n\t\t\t>>Vetor Z<<\n");
13      imprimeVetor(z, 0, tamVetor-1);
14
15          return *z;
16  }
```

Por fim, é gerado o vetor criptografado Z, que é resultado do XOR (ou exclusivo) entre X e Y.

```
1   int comprimeDados(int val1, int val2){
2       if(val1 == 0 && val2 == 0)
3           return 0;
4       else if(val1 == 1 && val2 == 1)
5           return 1;
6       else if(val1 == 0 && val2 == 1)
7           return 2;
8       else if(val1 == 1 && val2 == 0)
9           return 3;
10      else{
11          printf("Error"); exit(0);
12      }
13  }
14
15  int descomprimeDados(int val1){
16      if(val1 == 0)
17          return 00;
18      else if(val1 == 1)
19          return 11;
20      else if(val1 == 2)
21          return 01;
22      else if(val1 == 3)
23          return 10;
24      else{
25          printf("Error"); exit(0);
26      }
27  }
```

Aqui é onde ocorre a designação dos valores para a compactação (onde a configuração de 2 bits resulta em 1 bit) e também para a descompactação (onde 1 bit retorna 2 bits), conforme tabelas 1 e 2.

```
1   //RESULTADO DA COMPACTACAO
2   int compactacao(int *compacta, int tamVetor, int *z){
3       int i, j = 0;
4
5       for(i = 0; i < tamVetor; i+=2){
6           compacta[j] = comprimeDados(z[i], z[i+1]);
```

```
7           j++;
8       }
9
10      printf("\n\n\t\t\t>>Vetor Compactado<<\n");
11      imprimeVetor(compacta, 0, ((tamVetor-1)/2));
12      return *compacta;
13  }
14
15  //RESULTADO DA DESCOMPACTACAO
16  int descompactacao(int *descompacta, int tamVetor, int *compacta)
        {
17      int i, j = 0;
18
19      for(i = 0; i < tamVetor; i++){
20          descompacta[j] = descomprimeDados(compacta[i]);
21          j++;
22      }
23
24      printf("\n\t\t\t>>Vetor Descompactado<<\n");
25      imprimeVetor(descompacta, 0, ((tamVetor-1)/2));
26      return *descompacta;
27  }
```

Finalizando o algoritmo, neste passo é onde temos a aplicação do compactador e descompactador. A compactação é baseada no vetor Z e a descompactação é baseada no resultado da compactação.

# 4 Conclusão

Após fazermos as experimentações exatamente conforme descritas no artigo, podemos concluir que a técnica proposta realmente funciona e traz resultados eficientes e satisfatórios. Além disso, a implementação e o entendimento do mesmo é bem simples, o que o torna bem acessível e praticável.

# Referências

[Mahmud, 2012] Mahmud, Salauddin. 2012. An improved data compression method for general data. *International Journal of Scientific & Engineering Research*, **3**(3), 2.