

Compressão de texto

A compressão de texto consiste em representar o texto original de documentos em menos espaço. Deve-se substituir os símbolos do texto por outros que ocupam um número menor de *bits* ou *bytes*.

Ganho obtido: o texto comprimido ocupa menos espaço de armazenamento, levando menos tempo para ser pesquisado e para ser lido do disco ou transmitido por um canal de comunicação.

Preço a pagar: custo computacional para codificar e decodificar o texto.

Além da economia de espaço, outros aspectos relevantes são:

Velocidade de compressão e descompressão (Em muitas situações, a velocidade de descompressão é mais importante que a compressão);

Possibilidade de realizar casamento de cadeias diretamente no texto comprimido (A busca sequencial da cadeia comprimida pode ser bem mais eficiente do que descomprimir o texto a ser pesquisado);

Acesso direto a qualquer parte do texto comprimido, possibilitando o início da descompressão a partir da parte acessada (Um sistema de recuperação de informações para grandes coleções de documentos que estejam comprimidos necessita acesso direto a qualquer ponto do texto comprimido);

Razão de compressão corresponde à porcentagem que o arquivo comprimido representa em relação ao tamanho do arquivo não comprimido, sendo utilizada para medir o ganho em espaço obtido por um método de compressão. Ex.: se o arquivo não comprimido possui 100 *bytes* e o arquivo comprimido possui 30 *bytes*, a razão é de 30%.

Huffman

Um método de codificação bem conhecido e utilizado é o de Huffman, proposto em 1952. Um código único, de tamanho variável, é atribuído a cada símbolo diferente do texto, códigos mais curtos são atribuídos a símbolos com frequência mais altas. *As implementações tradicionais do método de Huffman consideram caracteres como símbolos.

Para atender as necessidades dos sistemas de RI, deve-se considerar palavras como símbolos a serem codificados. *Métodos de Huffman baseados em caracteres e em palavras comprimem o texto para cerca de 60% e 25%, respectivamente.

Compressão de Huffman Usando palavras

Corresponde à técnica de compressão mais eficaz para textos em linguagem natural. Inicialmente, considera cada palavra diferente do texto como um símbolo, contando suas frequências e gerando um código de Huffman para as mesmas, a seguir, comprime o texto substituindo cada palavra pelo seu

código correspondente. *A tabela de símbolos do codificador é exatamente o vocabulário do texto, o que permite uma integração natural entre o método de compressão e arquivo invertido (sistema de RI).

A compressão é realizada em duas passadas sobre o texto:

Obtenção da frequência de cada palavra diferente.

Realização da compressão.

Um texto em linguagem natural é constituído de palavras e de separadores (caracteres que aparece, entre palavras, como espaço, vírgula, ponto, etc).

Uma forma eficiente de lidar com palavras e separadores é representar o espaço simples de forma implícita no texto comprimido. Se uma palavra é seguida de um espaço, somente a palavra é codificada; caso contrário, a palavra e o separador são codificados separadamente. No momento da decodificação, supõe-se que um espaço simples segue cada palavra, a não ser que o próximo símbolo corresponda a um separador.

Árvore de codificação

O algoritmo de Huffman constrói uma árvore de codificação, partindo-se de baixo para cima.

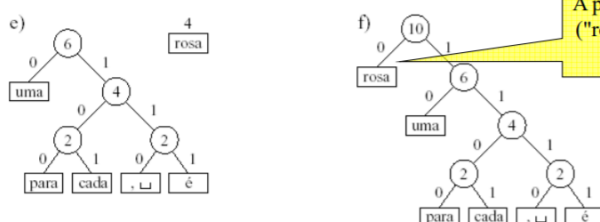
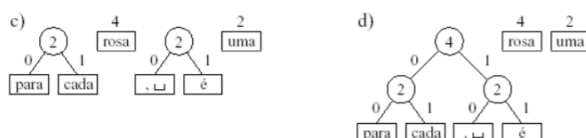
Inicialmente, há um conjunto de N folhas representando as palavras do vocabulário e suas respectivas frequências;

A cada intercalação, as duas árvores com as menores frequências são combinadas em uma única árvore e a soma de suas frequências é associada ao nó raiz da árvore gerada;

Ao final de (n-1) iterações, obtém-se a árvore de codificação, na qual o código associado a uma palavra é representado pela sequência dos rótulos das arestas da raiz à folha que a representa.

Árvore de codificação para o texto:

"para cada rosa, uma rosa é uma rosa"



A palavra mais frequente ("rosa") recebe o código mais curto ("0").

Ex.:

O método de Huffman produz a árvore de codificação que minimiza o comprimento do arquivo.

Existem várias árvores que produzem a mesma compressão, trocar o filho à esquerda de um nó por um filho à direita leva a uma árvore de codificação alternativa com a mesma razão de compressão.

A escolha preferencial é a árvore canônica. *Uma árvore de Huffman é canônica quando a altura da subárvore à direita de qualquer nó nunca é menor que a altura da subárvore à esquerda.

A representação do código por meio de uma árvore canônica de codificação facilita a visualização e sugere métodos triviais de codificação e decodificação.

Codificação: a árvore é percorrida emitindo *bits* ao longo de suas arestas;

Decodificação: os *bits* de entrada são usados para selecionar as arestas.

Essa abordagem é ineficiente tanto em termos de espaço quanto em termos de tempo.

Algoritmo de Moffat e Katajainen

O algoritmo criado em 1995, baseado na codificação canônica, apresenta comportamento linear em tempo e em espaço.

O algoritmo calcula os comprimentos dos códigos em lugar dos códigos propriamente ditos. *A compressão atingida é a mesma, independentemente dos códigos utilizados

Após o cálculo dos comprimentos, há uma forma elegante e eficiente para a codificação e a decodificação.

A entrada do algoritmo é um vetor A contendo as frequências das palavras em ordem decrescente.

Para o texto “Para cada rosa rosa, uma rosa é uma rosa”, o vetor A é:

4	2	1	1	1	1
---	---	---	---	---	---

4 – Rosa;

2 - Uma;

1 – Para;

1 – Cada;

1 – ‘,’;

1 – é.

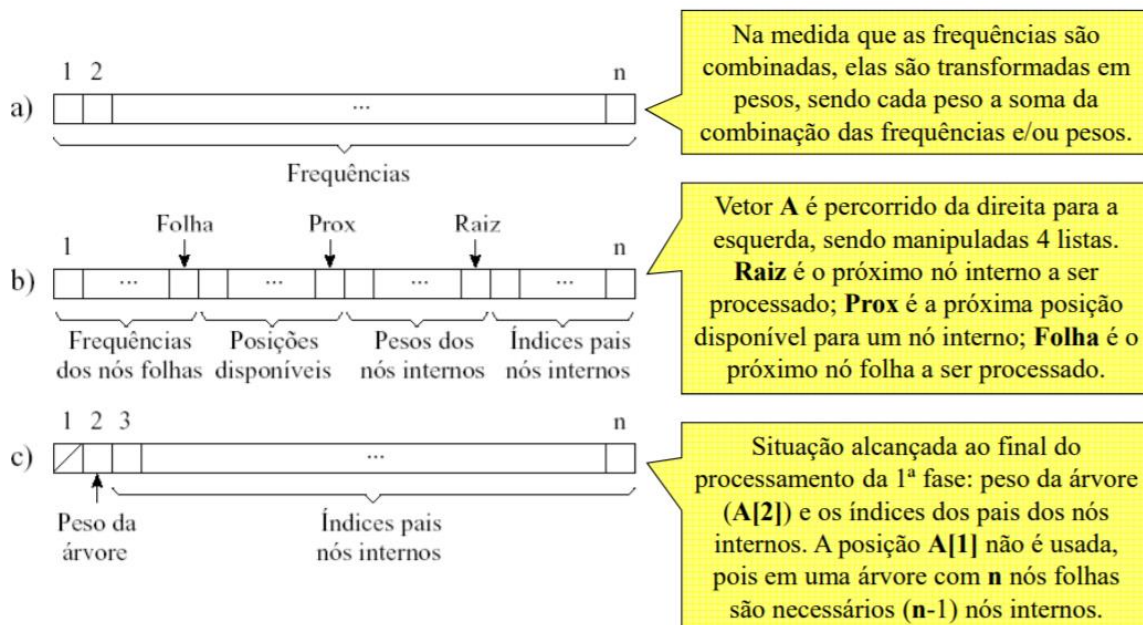
Durante a execução, são usados vetores logicamente distintos, mas que coexistem no mesmo vetor A.

O algoritmo divide-se em três fases distintas:

- 1 – Combinação dos nós;
- 2 – Determinação das profundidades dos nós internos;
- 3 – Determinação das profundidades dos nós folhas (comprimento dos códigos).

Exemplificação.

Primeira fase do algoritmo: Combinação dos nós



Código:

PrimeiraFase (A, n)

{ Raiz = n; Folha = n;

for (Prox = n; n >= 2; Prox--)

{ /* Procura Posicao */

if ((nao existe Folha) || ((Raiz > Prox) && (A[Raiz] <= A[Folha])))

{ A[Prox] = A[Raiz]; A[Raiz] = Prox; Raiz = Raiz - 1; /* No interno */ }

else { A[Prox] = A[Folha]; Folha = Folha - 1; /* No folha */ }

/* Atualiza Frequencias */

if ((nao existe Folha) || ((Raiz > Prox) && (A[Raiz] <= A[Folha])))

{ /* No interno */

A[Prox] = A[Prox] + A[Raiz]; A[Raiz] = Prox; Raiz = Raiz - 1;

}

else { A[Prox] = A[Prox] + A[Folha]; Folha = Folha - 1; /* No folha */ }

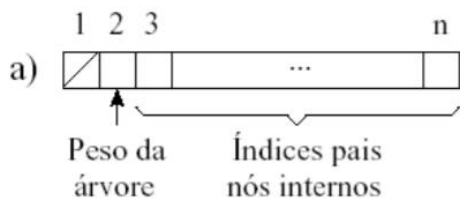
}

}

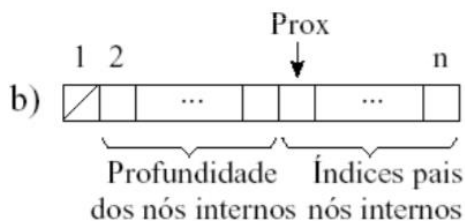
Exemplo da primeira fase do algoritmo:

	1	2	3	4	5	6	Prox	Raiz	Folha
a)	4	2	1	1	1	1	6	6	6
b)	4	2	1	1	1	1	6	6	5
c)	4	2	1	1	1	2	5	6	4
d)	4	2	1	1	1	2	5	6	3
e)	4	2	1	1	2	2	4	6	2
f)	4	2	1	2	2	4	4	5	2
g)	4	2	1	4	4	4	3	4	2
h)	4	2	2	4	4	4	3	4	1
i)	4	2	6	3	4	4	2	3	1
j)	4	4	6	3	4	4	2	3	0
k)	10	2	3	4	4		1	2	0

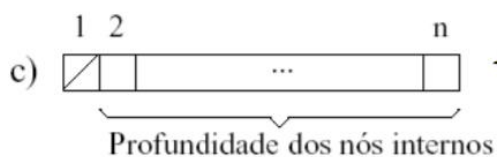
Segunda fase: profundidade dos nós internos



Resultado da 1ª fase. Vetor A é convertido, da esquerda para a direita, na profundidade dos nós internos.



Prox é o próximo índice de pai dos nodos internos a ser processado. **A[2]** representa a raiz da árvore. Chega-se ao desejado (profundidade dos nós internos), fazendo **A[2] = 0** e **A[Prox] = A[A[prox]] + 1** (uma unidade maior que seu pai), com **Prox** variando de 3 até **n**.



Situação alcançada ao final do processamento da 2ª fase: profundidade dos nós internos. A posição **A[1]** não é usada, pois em uma árvore com **n** nós folhas são necessários **(n-1)** nós internos.

Código:

SegundaFase (A, n)

{ A[2] = 0;

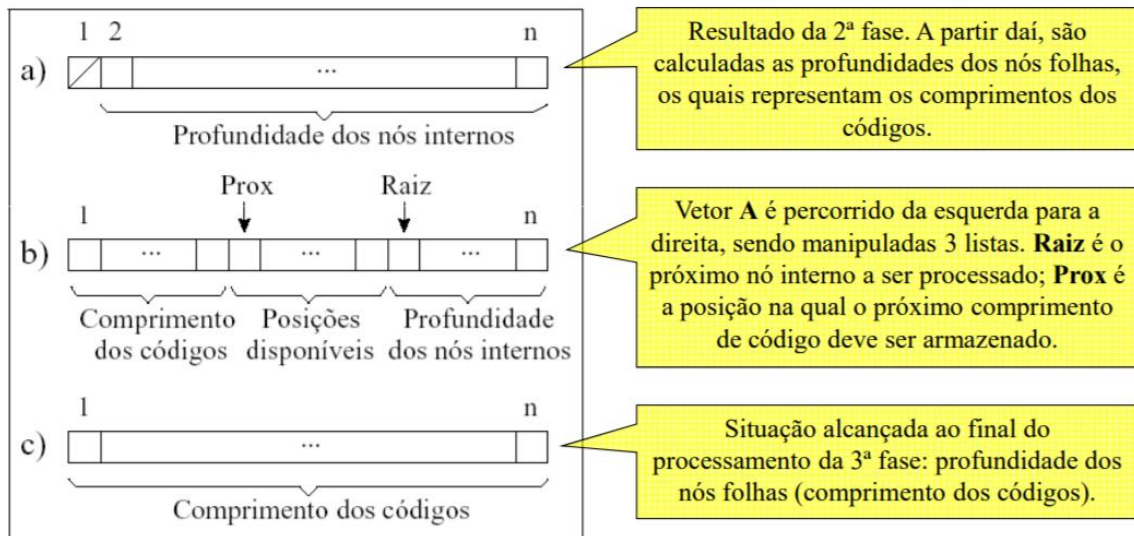
for (Prox = 3; Prox <= n; Prox++) A[Prox] = A[A[Prox]] + 1;

}

Resultado:

10	0	1	2	3	3
----	---	---	---	---	---

Terceira fase: Profundidade dos nós folhas



Código:

TerceiraFase (A, n)

```
{ Disp = 1; u = 0; h = 0; Raiz = 2; Prox = 1;
  while (Disp > 0)
  { while (Raiz <= n && A[Raiz] == h) { u = u + 1; Raiz = Raiz + 1; }
    while (Disp > u) { A[Prox] = h; Prox = Prox + 1; Disp = Disp - 1; }
    Disp = 2 * u; h = h + 1; u = 0;
  }
}
```

Disp armazena quantos nós estão disponíveis no nível **h** da árvore.
u indica quantos nós do nível **h** são internos.

Resultado:

1	2	4	4	4	4
---	---	---	---	---	---

Programa completo para calcular o comprimento dos códigos a partir de um vetor de frequências:

```
CalculaCompCodigo(A, n){
    A = PrimeiraFase (A, n);
    A = SegundaFase (A, n);
    A = TerceiraFase (A, n);
}
```

Obtenção dos Códigos Canônicos

As propriedades dos códigos canônicos são:

Os comprimentos dos códigos seguem o algoritmo de Huffman;

Códigos de mesmo comprimento são inteiros consecutivos.

A partir dos comprimentos pelo algoritmo de Moffat e Katajainen, o cálculo dos códigos é simples:

O primeiro código é composto apenas por zeros;

Para os demais, adiciona-se 1 ao código anterior e faz-se um deslocamento à esquerda para obter-se o comprimento adequado quando necessário;

Ex.:

i	Símbolo	Código Canônico
1	rosa	0
2	uma	10
3	para	1100
4	cada	1101
5	,□	1110
6	é	1111

Codificação e Decodificação

Os algoritmos são baseados no fato: Códigos de mesmo comprimento são inteiros consecutivos.

Os algoritmos usam dois vetores com MaxCompCod (o comprimento do maior código) elementos:

Base: Indica, para um dado comprimento c , o valor inteiro do 1º código com tal comprimento;

$$\text{Base}[c] = \begin{cases} 0 & \text{se } c = 1, \\ 2 \times (\text{Base}[c-1] + w_{c-1}) & \text{caso contrário,} \end{cases}$$

Nº de códigos com comprimento (c-1).

Offset: indica, para um dado comprimento c , o índice no vocabulário da 1ª palavra de tal comprimento.

c	Base[c]	Offset[c]
1	0	1
2	2	2
3	6	2
4	12	3

Código:

Codifica (Base, Offset, i , MaxCompCod)

{ $c = 1$;

while ($i \geq \text{Offset}[c + 1]$) && ($c + 1 \leq \text{MaxCompCod}$)

$c = c + 1$;

Codigo = $i - \text{Offset}[c] + \text{Base}[c]$;

}

Parâmetros: vetores **Base** e **Offset**, índice i do símbolo a ser codificado e o comprimento **MaxCompCod** dos vetores.

Cálculo do comprimento c de código a ser utilizado.

O código corresponde à soma da ordem do código para o comprimento c ($i - \text{Offset}[c]$) com o valor inteiro do 1º código de comprimento c (**Base**[c]).

Para $i = 4$ ("cada"), calcula-se que seu código possui comprimento 4 e verifica-se que é o 2º código de tal comprimento. Assim, seu código é $13(4 - \text{offset}[4] + \text{base}[4])$: 1101

Parâmetros: vetores **Base** e **Offset**, o arquivo comprimido e o comprimento **MaxCompCod** dos vetores.

Decodifica (Base, Offset, ArqComprimido, MaxCompCod)

{ $c = 1$;

Codigo = LeBit (ArqComprimido);

Identifica o código a partir de uma posição do arquivo comprimido.

while (((Codigo << 1) >= Base[$c + 1$]) && ($c + 1 \leq \text{MaxCompCod}$))

{ Codigo = (Codigo << 1) || LeBit (ArqComprimido);

$c = c + 1$;

}

$i = \text{Codigo} - \text{Base}[c] + \text{Offset}[c]$;

}

O arquivo de entrada é lido *bit-a-bit*, adicionando-se os *bits* lidos ao código e comparando-o com o vetor **Base**.

Código de compressão:

1ª Etapa: o arquivo texto é percorrido e o vocabulário é gerado juntamente com a frequência de cada palavra.

Uma tabela *hash* com tratamento de colisão é utilizada para que as operações de inserção e pesquisa no vetor de vocabulário sejam realizadas com custo $O(1)$.

2ª Etapa:

O vetor vocabulário é ordenado pelas frequências de suas palavras;

Calcula-se o comprimento dos códigos (algoritmo de Moffat e Katajainen);

Os vetores *Base*, *Offset* e *Vocabulário* são construídos e gravados no início do arquivo comprimido;

A tabela *hash* é reconstruída a partir do vocabulário no disco, como preparação para a 3ª Etapa.

3ª Etapa:

O arquivo texto é novamente percorrido;

As palavras são extraídas e codificadas;

Os códigos correspondentes são gravados no arquivo comprimido.

Compressao (ArqTexto, ArqComprimido)

{ */* Primeira etapa */*

while (!feof (ArqTexto))

 { Palavra = ExtraiProximaPalavra (ArqTexto);

 Pos = Pesquisa (Palavra, Vocabulario);

if Pos é uma posicao valida

 Vocabulario[Pos].Freq = Vocabulario[Pos].Freq + 1

else Insere (Palavra, Vocabulario);

 }

/ Segunda etapa */*

Vocabulario = OrdenaPorFrequencia (Vocabulario);

Vocabulario = CalculaCompCodigo (Vocabulario, n);

ConstroiVetores (Base, Offset, ArqComprimido);

Grava (Vocabulario, ArqComprimido);

LeVocabulario (Vocabulario, ArqComprimido);

```

/* Terceira etapa */
PosicionaPrimeiraPosicao (ArqTexto);
while (!feof(ArqTexto))
{
    Palavra = ExtraiProximaPalavra (ArqTexto);
    Pos = Pesquisa (Palavra, Vocabulario);
    Codigo = Codifica (Base, Offset,
                      Vocabulario[Pos].Ordem, MaxCompCod);
    Escreve (ArqComprimido, Codigo);
}
}

```

Código de descompressão:

O processo de descompressão é mais simples do que o de compressão:

Leitura dos vetores *Base*, *Offset* e *Vocabulário* gravados no início do arquivo comprimido;

Leitura dos códigos do arquivo comprimido, decodificando-se e gravando as palavras correspondentes no arquivo texto.

```

Descompressao (ArqTexto, ArqComprimido)
{
    LerVetores (Base, Offset, ArqComprimido);
    LeVocabulario (Vocabulario, ArqComprimido);
    while (!feof(ArqComprimido))
    {
        i = Decodifica (Base, Offset, ArqComprimido, MaxCompCod);
        Grava (Vocabulario[i], ArqTexto);
    }
}

```
