Fábio Henrique Alves Fernandes - 19.1.4128

Resumo: Compressão de Texto

Compressão de Texto

A ideia da compressão de textos é reduzir o tamanho de um texto, para fazê-lo gastar menos espaço, substituindo símbolos do texto por outros que ocupam um número menor de bytes.Um texto comprimido, por ocupar bem menos espaço, leva menos tempo para ser pesquisado, para ser lido do disco, ou até mesmo transmitido por um canal de comunicação, porém, tem seu custo computacional para a codificação e a decodificação. Uma das outras vantagens de se comprimir o texto, também é vista para o casamento de cadeias, onde a busca sequencial pode ser bem mais eficiente. Também temos acesso direto a qualquer parte do texto comprimido, possibilitando o início da descompressão a partir da parte acessada.

A 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%.

Compressão de Huffman

Compressão de Huffman é um método de compressão bem conhecido e utilizado. 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 Recuperação de Informação, 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.

A Compressão de Huffman Usando palavras é um dos métodos de compressão mais eficaz para textos em linguagem natural. Inicialmente, considera cada palavra diferente do

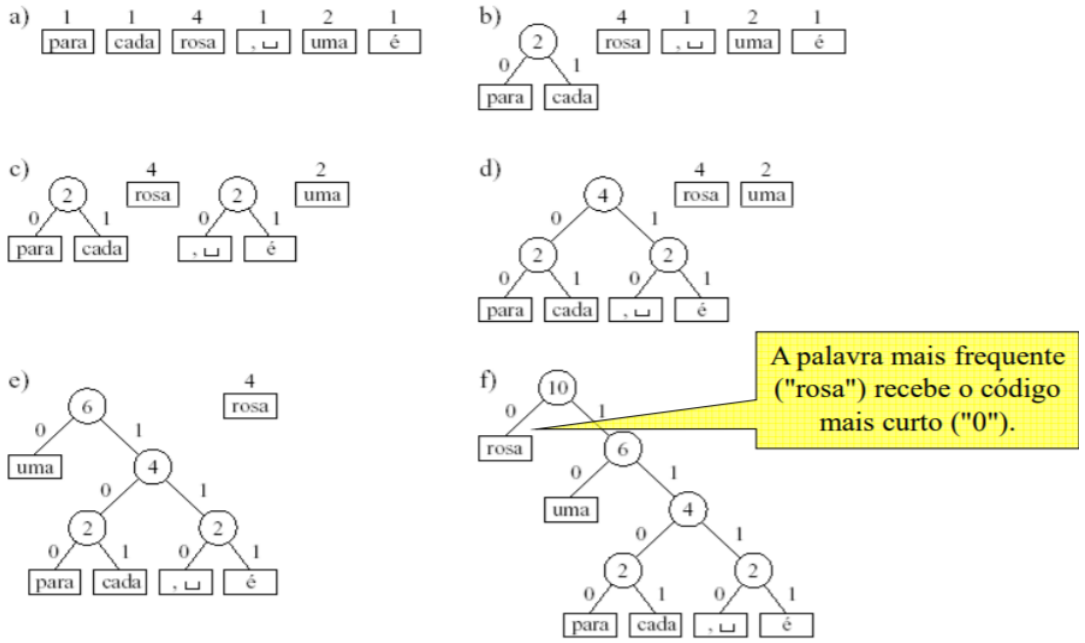
texto como um símbolo, contando suas frequências e gerando um código de Huffman para as mesmas, a seguir, comprime o texto substituindo cada palavra pelo seu código correspondente. A tabela de símbolos do codificador é exatamente o vocabulário do texto, o que permite uma integração natural entre o método de compressão e arquivo invertido. Um texto em linguagem natural é constituído de palavras e de separadores (caracteres que aparecem entre palavras, como espaço, vírgula, ponto, etc). A compressão é realizada em duas passadas sobre o texto: para obtenção da frequência de cada palavra diferente e para a realização da compressão.

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.

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.

Exemplo de uma árvore construída pela algoritmo:

Árvore de codificação para o texto:
"para cada rosa rosa, uma rosa é uma rosa"



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 a 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. Na codificação, a árvore é percorrida emitindo bits ao longo de suas arestas. Já na 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 de Moffat e Katajainen, criado em 1995, baseado em árvore canônica, é um algoritmo com um comportamento linear em tempo e espaço, onde faz-se o cálculo dos comprimentos dos códigos ao invés dos códigos propriamente ditos. A compressão é a mesma do método de Huffman. 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

Primeira fase

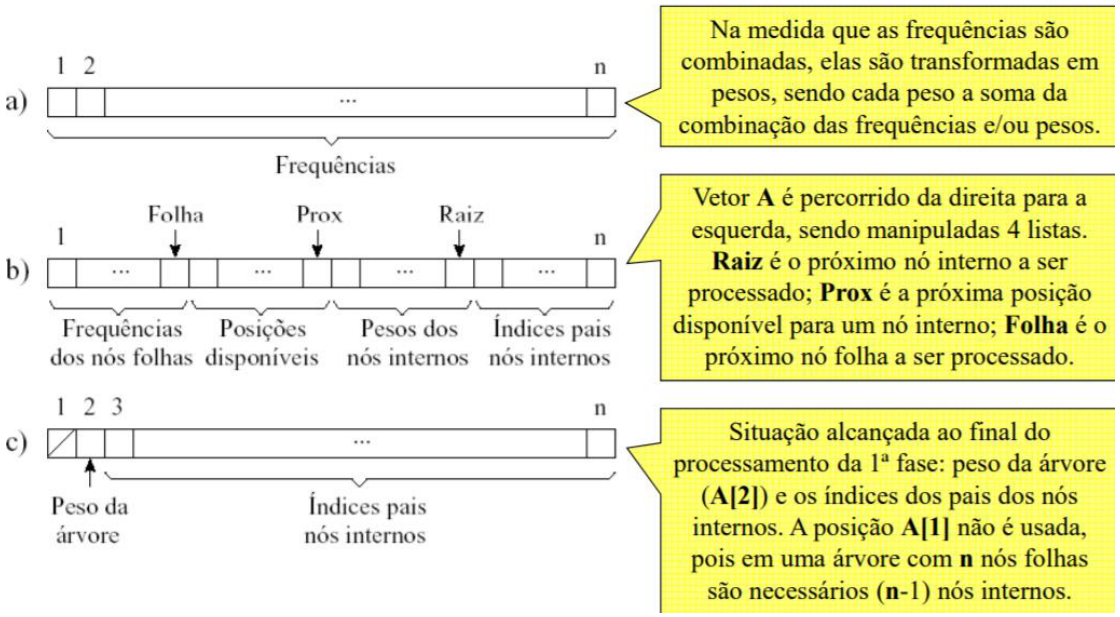
Algoritmo

```
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 */ }
  }
}
```

Execução

	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

Explicação



Segunda Fase

Algoritmo

```
SegundaFase (A, n)
{ A[2] = 0;
   for (Prox = 3; Prox <= n; Prox++) A[Prox] = A[A[Prox]] + 1;
}
```

Resultado

	0	1	2	3	3

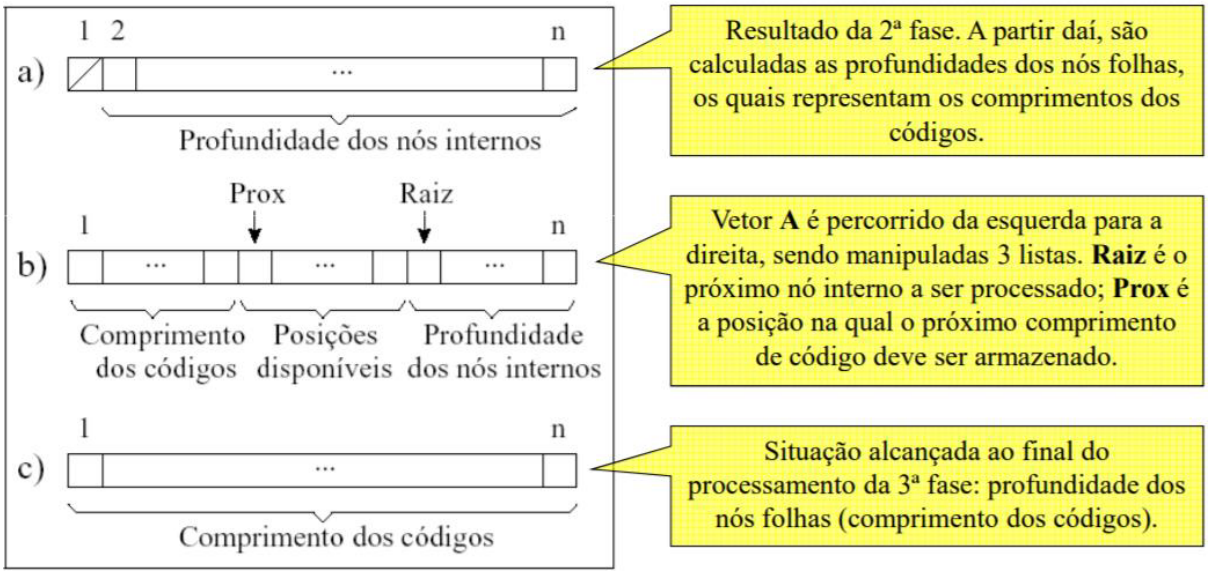
Terceira Fase

Algoritmo

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

```
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;
  }
}
```

Explicação



Resultado final dos comprimentos

1	2	4	4	4	4

Para a obtenção dos Códigos Canônicos, usamos as seguintes propriedades: 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.

Usando o mesmo exemplo anterior, podemos exemplificar:

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 de que códigos de mesmo comprimento são inteiros consecutivos. Os algoritmos usam dois vetores com o tamanho do comprimento do maior código (MaxCompMod).

A Base indica, para um dado comprimento ‘c’, o valor inteiro do 1º código com tal comprimento.

Nº de códigos com comprimento (**c-1**).

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

Já o 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

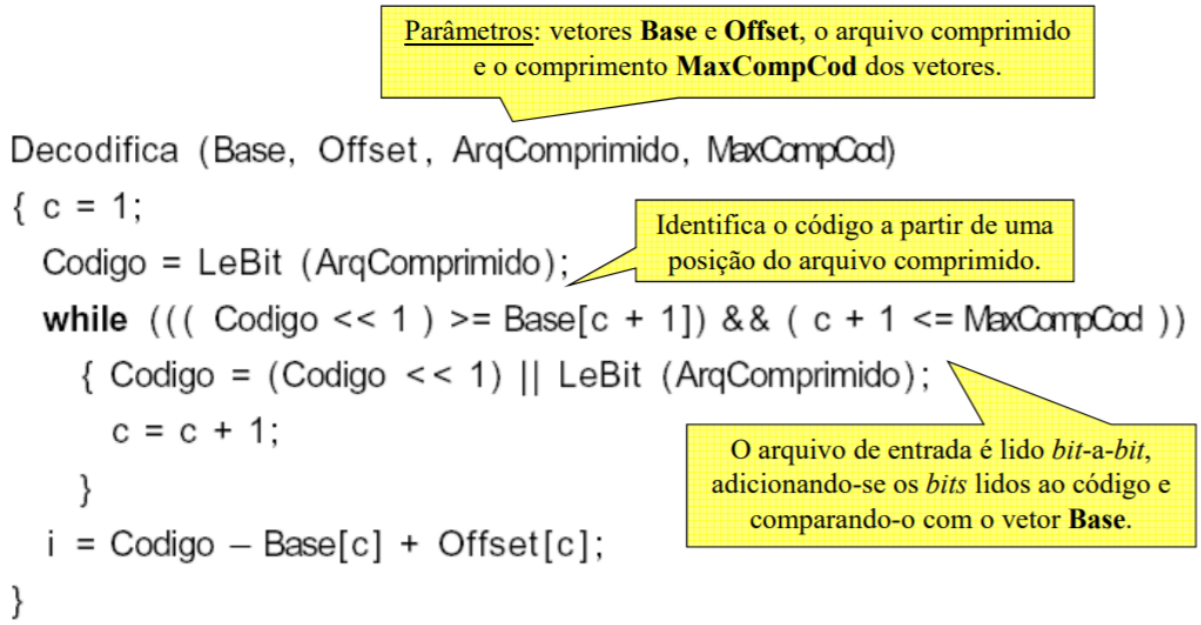
```
Codifica (Base, Offset, i, MaxCompCod)
{ c = 1;
  while ( i >= Offset[c + 1] ) && (c + 1 <= MaxCompCod )
    c = c + 1;
  Codigo = i – Offset[c] + Base[c];
}
```

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 – Offset[c]**) com o valor inteiro do 1º código de comprimento **c** (**Base[c]**).

Como exemplo, 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 – offset [4] + base [4]): 1101



Para o processo de compressão, 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). 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. Depois, o arquivo texto é novamente percorrido, as palavras são extraídas e codificadas e 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);
    }
}
```

Já a descompressão é bem mais simples. Lê se os vetores Base, Offset e Vocabulário gravados no início do arquivo comprimido, também os códigos do arquivo comprimido, fazendo a decodificação 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);
    }
}
```