Compressão de Textos

Estrutura de Dados II (BCC203)

Prof. Guilherme Tavares de Assis

Universidade Federal de Ouro Preto – UFOP

Instituto de Ciências Exatas e Biológicas – ICEB

Departamento de Computação – DECOM

Introdução

- 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.

Introdução

- Além da economia de espaço, outros aspectos relevantes são:
  - Velocidade de compressão e de descompressão.
    - Em muitas situações, a velocidade de descompressão é mais importante que a de compressão (ex.: bancos de dados textuais).
  - 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 qq parte do texto comprimido, possibilitando o início da descompressão a partir da parte acessada.
    - Um sistema de recuperação de informação para grandes coleções de documentos que estejam comprimidos necessita acesso direto a qualquer ponto do texto comprimido.

Introdução

- Razão de compressão corresponde à porcentagem que o arquivo comprimido representa em relação ao tamanho do arquivo não comprimido.
  - É 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%.

Introdução

- 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 frequências 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 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 (sistemas de RI).
  - A seguir, comprime o texto substituindo cada palavra pelo seu código correspondente.

- A compressão é realizada em duas passadas sobre o texto:
  - Obtenção da frequência de cada palavra diferente.
  - Realização da compressão.

Compressão de Huffman Usando Palavras

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

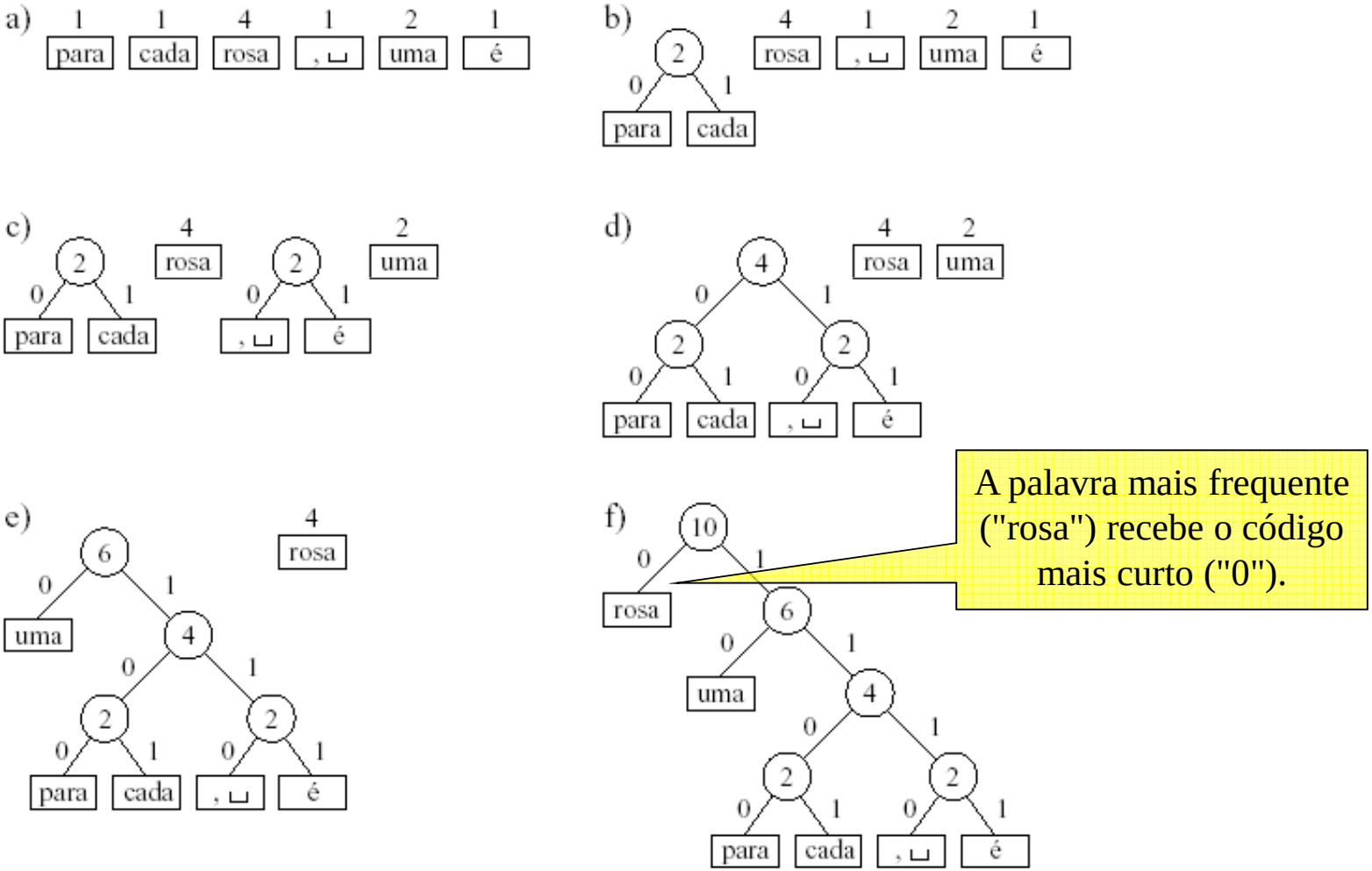
- Uma forma eficiente de lidar com palavras e separadores é representar o espaço simples de forma implícita no texto comprimido.
  - Se uma palavra é seguida de um espaço, somente a palavra é codificada; caso contrário, a palavra e o separador são codificados separadamente.
  - No momento da decodificação, 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 interação, 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

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



Árvore de Codificação

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

- 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.

Árvore de Codificação

- 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 de Moffat e Katajainen (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.

Algoritmo de Moffat e Katajainen

- 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:

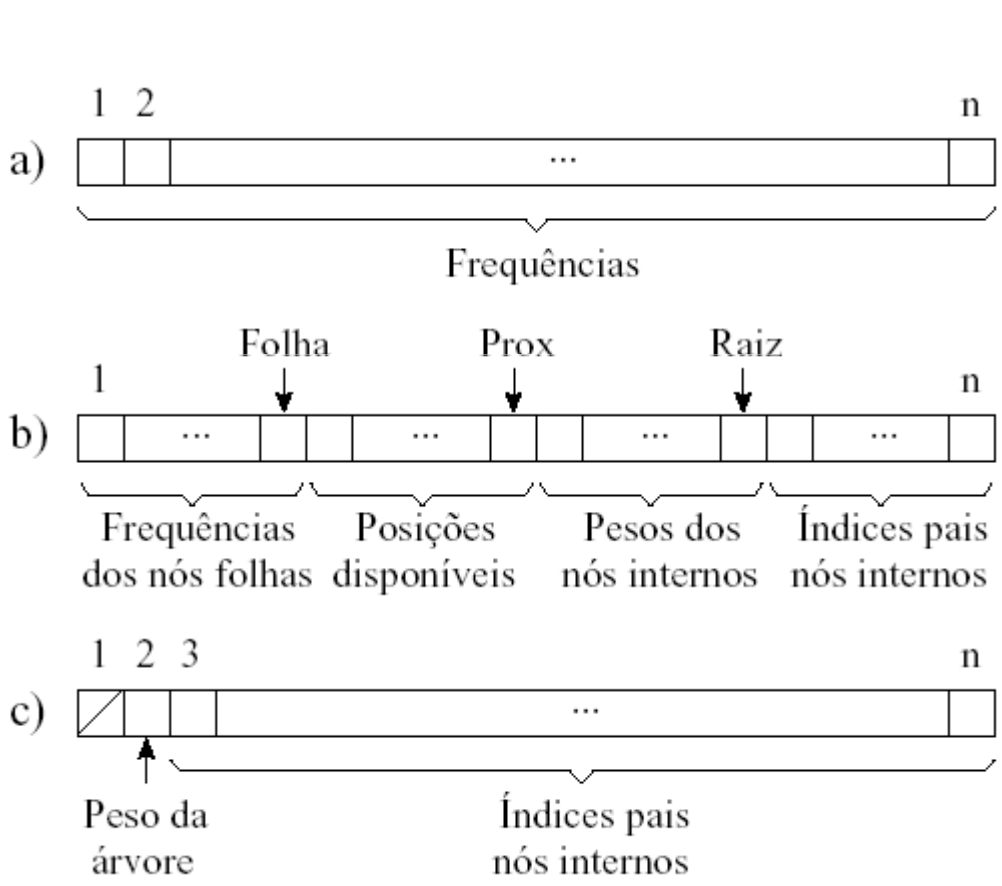
4	2	1	1	1	1

  - 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 (comprimentos dos códigos).

Algoritmo de Moffat e Katajainen

- Primeira fase do algoritmo: combinação dos nós.



Na medida que as frequências são combinadas, elas são transformadas em pesos, sendo cada peso a soma da combinação das frequências e/ou pesos.

Vetor **A** é percorrido da direita para a esquerda, sendo manipuladas 4 listas. **Raiz** é o próximo nó interno a ser processado; **Prox** é a próxima posição disponível para um nó interno; **Folha** é o próximo nó folha a ser processado.

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

Algoritmo de Moffat e Katajainen

```
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 */ }
  }
}
```

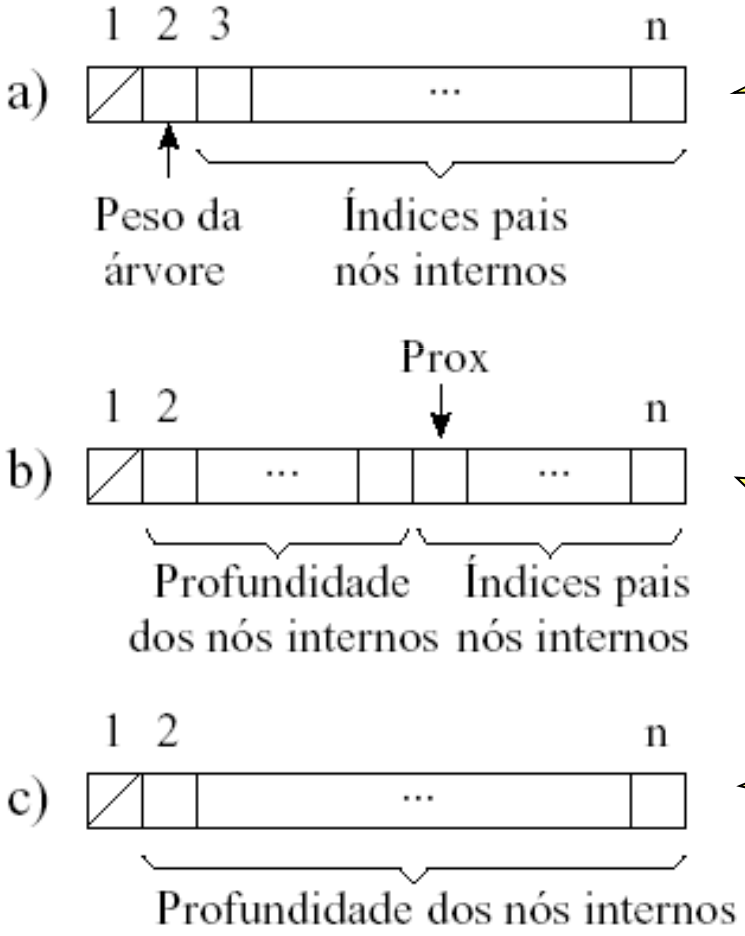
Algoritmo de Moffat e Katajainen

- 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

Algoritmo de Moffat e Katajainen

- Segunda fase do algoritmo: 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.

Algoritmo de Moffat e Katajainen

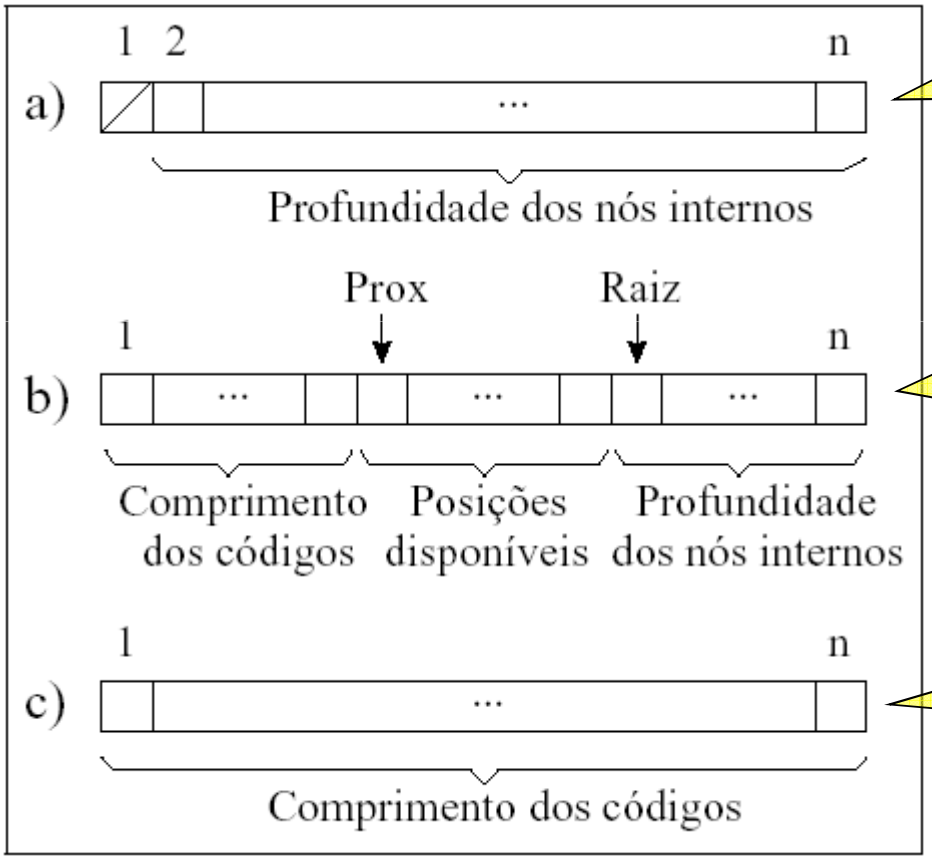
```
SegundaFase (A, n)
{ A[2] = 0;
  for (Prox = 3; Prox <= n; Prox++) A[Prox] = A[A[Prox]] + 1;
}
```

- Resultado da segunda fase:

	0	1	2	3	3

Algoritmo de Moffat e Katajainen

- Terceira fase do algoritmo: profundidade dos nós folhas.



Resultado da 2ª fase. A partir daí, são calculadas as profundidades dos nós folhas, os quais representam os comprimentos dos códigos.

Vetor **A** é percorrido da esquerda para a direita, sendo manipuladas 3 listas. **Raiz** é o próximo nó interno a ser processado; **Prox** é a posição na qual o próximo comprimento de código deve ser armazenado.

Situação alcançada ao final do processamento da 3ª fase: profundidade dos nós folhas (comprimento dos códigos).

Algoritmo de Moffat e Katajainen

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;
  }
}
```

- Resultado da terceira fase:

1	2	4	4	4	4

Algoritmo de Moffat e Katajainen

- 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 do 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 obtidos 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.

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.

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

- 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;

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

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

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

Codificação e Decodificação

```
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]**).

- 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.

Codificação e Decodificação

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

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

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

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

Codificação e Decodificação

Decodificação da sequência de *bits* "1101":

c	LeBit	Codigo	Codigo $<< 1$	Base[$c+1$]
1	1	1	-	-
2	1	10 **or** $1 = 11$	10	10
3	0	110 **or** $0 = 110$	110	110
4	1	1100 **or** $1 = 1101$	1100	1100

- A 1ª linha da tabela é o estado inicial do *while*, quando já foi lido o primeiro *bit* da sequência, atribuído à *Codigo*.
- As linha seguintes representam a situação do anel *while* após cada respectiva iteração.
  - Na linha dois, o segundo *bit* foi lido (*bit* 1) e *Codigo* recebe o código anterior deslocado à esquerda de um *bit* seguido da operação *or* com o bit lido.
- De posse do código, *Base* e *Offset* são usados para identificar o índice i da palavra no vocabulário:

$$i = Codigo - Base[c] + Offset[c]$$

Compressão

```
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);
    }
```

- O processo de compressão é realizado em 3 etapas.
- Na 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).

Compressão

```
/* Segunda etapa */
Vocabulario = OrdenaPorFrequencia (Vocabulario);
Vocabulario = CalculaCompCodigo (Vocabulario, n);
ConstroiVetores (Base, Offset, ArqComprimido);
Grava (Vocabulario, ArqComprimido);
LeVocabulario (Vocabulario, ArqComprimido);
```

- Na 2ª etapa:
  - o vetor Vocabulario é ordenado pelas frequências de suas palavras;
  - calcula-se o comprimento dos códigos (algoritmo de Moffat e Katajainen);
  - os vetores *Base*, *Offset* e *Vocabulario* são construídos e gravados no início do arquivo comprimido;
  - a tabela *hash* é reconstruída a partir da leitura do vocabulário no disco, como preparação para a 3ª etapa.

Compressão

```
/* 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);
  }
}
```

- Na 3ª etapa:
  - o arquivo texto é novamente percorrido;
  - as palavras são extraídas e codificadas;
  - os códigos correspondentes são gravados no arquivo comprimido.

Decompressão

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

- O processo de decompressão é mais simples do que o de compressão:
  - Leitura dos vetores *Base*, *Offset* e *Vocabulario* gravados no ínicio do arquivo comprimido.
  - Leitura dos códigos do arquivo comprimido, descodificando-os e gravando as palavras correspondentes no arquivo texto.