Métodos de Compressão

Organização e Recuperação de Dados

Profa. Valéria

UEM – CTC – DIN

Slides preparados com base no Cap. 5 do livro FOLK, M.J. & ZOELLICK, B. *File Structures*. 2nd Edition, Addison-Wesley Publishing Company, 1992, no Cap. 5 do livro SEDGEWICK, R. & WAYNE, K. *Algorithms*, 4th Edition, Addison-Wesley, 2011 e nos slides disponibilizados pelo Prof. Pedro de Azevedo Berger (DCC/UnB)

Compressão de dados

- Por que tornar os arquivos menores?
  - Ocupam menos espaço, ficando mais baratos
  - Podem ser transmitidos mais rapidamente
- Algumas técnicas de compressão são gerais e outras são específicas para certos tipos de dados, como áudio e imagem
- A variedade de técnicas é enorme
  - Veremos alguns exemplos de técnicas gerais e sem perda

Compressão de dados

- O objetivo da compressão é reduzir a quantidade de bits necessária para representar os dados
- Deve preservar a informação original
  - Na compressão com perda, parte dos dados é perdida, mas a informação permanece
- Em teoria, a compressão consiste em detectar e eliminar informação redundante

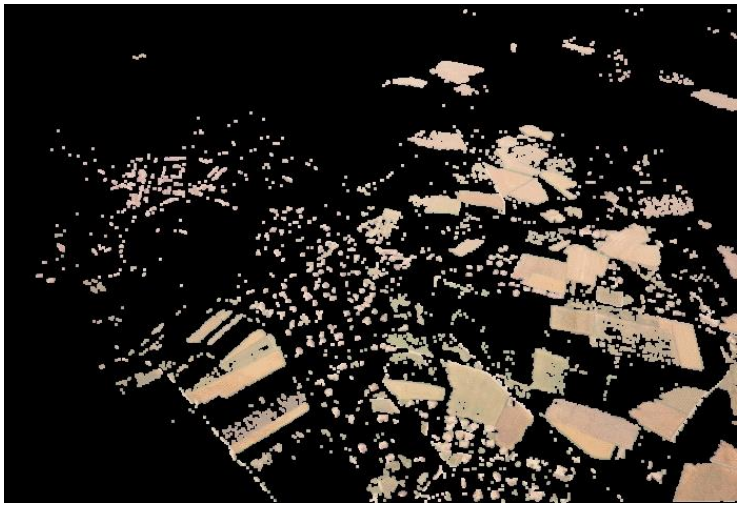
  ... porção de informação desnecessária e, sendo assim, repetida, que pode ser eliminada de forma que a informação original continua completa ou pelo menos pode ser recuperada.

  Shannon (1948)

RLE

- Um método simples de compressão é o *Run-length encoding* (RLE)
  - Indicado para arquivos nos quais sequências de bytes de mesmo valor são frequentes
  - Por exemplo, imagens segmentadas

(a) Original



(b) Segmentada

RLE

- Arquivos de imagem utilizam uma matriz na qual cada célula representa um pixel
  - A quantidade de bits utilizada para representar um pixel varia de acordo com o formato.
  - Por exemplo, um arquivo no formato BMP com 256 cores utiliza 8 bits por pixel
- Na imagem segmentada do slide anterior, existem várias sequências de pixels pretos, cada um representado pelo valor 0x00 → matriz esparsa
- Imagens que constituem matrizes esparsas são boas candidatas para a aplicação de compressão RLE

RLE

- Algoritmo
  1. Escolha um caractere especial para ser o marcador de repetição
  2. Leia a sequência original e copie os valores para nova sequência, exceto quando o mesmo valor ocorrer mais de k vezes consecutivas
  3. Quando o mesmo valor ocorrer mais de k vezes, substitua toda a sequência de repetição pela seguinte sequência de valores, nesta ordem:
     1. O código indicador de repetição
     2. O valor que se repete
     3. O número de vezes que o valor se repete na sequência original

RLE

- Exemplo
  - Suponha que o código hexadecimal **0xff** foi escolhido para ser o marcador de repetição e que $k = 3$

Sequência de bytes original (em hexa)
22 23 24 24 24 24 24 24 24 25 26 26 26 26 26 26 25 24 24

Sequência de bytes comprimida (em hexa)
22 23 **ff 24 07** 25 **ff 26 06** 25 24 24

  - 19 bytes (original) foram reduzidos para 12 bytes (comprimida)
  - Observe que a supressão de sequências ocorreu apenas quando um valor se repetiu mais de 3 vezes

RLE

- RLE pode ser utilizado com qualquer tipo de dados esparsos, como valores coletados de instrumentos e outras matrizes/vetores esparsos
- A compressão é garantida?
  - Não, depende dos dados que serão comprimidos e do fator de repetição (valor de k) utilizado

Códigos de comprimento variável

- Normalmente, utiliza-se códigos de tamanho fixo para a representação de arquivos
  - P.e., os símbolos da tabela ASCII são representados por códigos de um byte (8 bits)
- Mas é comum que certos símbolos ocorram com maior frequência do que outros
  - P.e., em um texto, as vogais são mais frequentes do que as consoantes
- Se utilizarmos códigos menores para os valores mais frequentes, teremos uma representação mais compacta

Códigos de comprimento variável

- Os códigos de comprimento variável são uma das técnicas mais antigas e comuns para compressão
- Exemplos de códigos de comprimento variável
  - Código Morse
    - A tabela de correspondência entre os símbolos e códigos é fixa
  - Código de Huffman
    - A tabela de correspondência entre os símbolos e códigos é dinâmica → é construída no processo de compactação e depende do arquivo sendo codificado

Código de Huffman

- Exemplo:

I		A	M		S	A	M	M	Y

Codificação ASCII
→10 bytes (80 bits)

Letra	**I**	**\b**	**A**	**M**	**S**	**Y**
Frequências	1	2	2	3	1	1
Códigos	**1010**	**00**	**01**	**11**	**1011**	**100**

- Mensagem codificada

1010000111001011011111100

Compactado
→ 25 bits

- Note que o código de Huffman é livre de prefixo → nenhum código é prefixo de outro

Código de Huffman

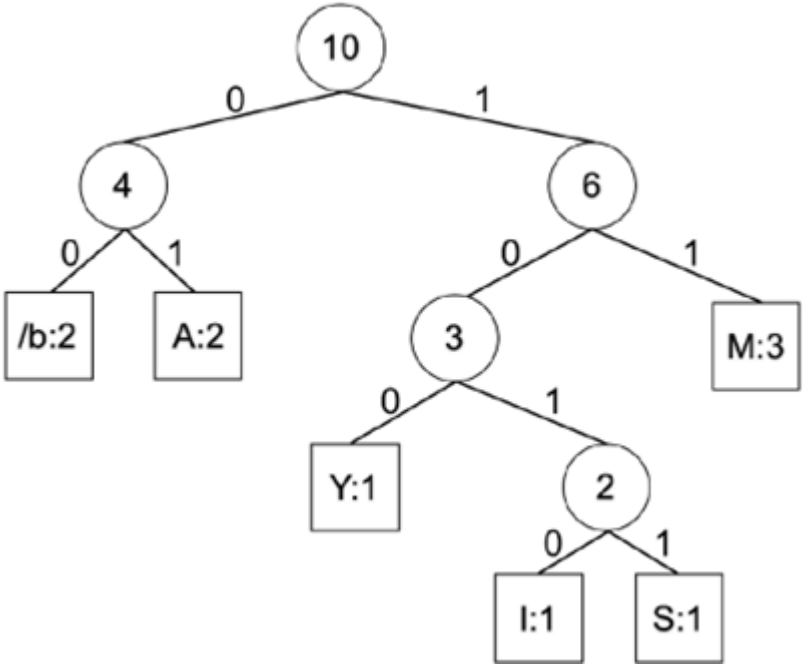
- Para a compactar um arquivo com código de Huffman:
  - Determine a frequência dos símbolos no arquivo
  - Construa uma árvore de Huffman
    - Nessa árvore, o caminho da raiz até a folha de um símbolo determina o código associado àquele símbolo
    - Os valores mais frequentes serão associados aos ramos mais curtos e os valores menos frequentes serão associados aos ramos mais longos
  - A partir da árvore de Huffman, gere a tabela de códigos e utilize-a para codificar os dados

Código de Huffman

- Exemplo:

I		A	M		S	A	M	M	Y

Símbolo	I	S	Y	\b	A	M
Frequência	1	1	1	2	2	3



- Cada folha representa um símbolo do alfabeto
- Os números nos nós internos representam a frequência acumulada pelos símbolos daquele ramo
- Os bits acumulados no caminho da raiz até uma folha compõem o código do símbolo

Símbolo	I	\b	A	M	S	Y
Código	1010	00	01	11	1011	100

Propriedades da árvore de Huffman

- Cada nó interno tem dois filhos
- As menores frequências estão mais distantes da raiz
- As duas menores frequências são nós irmãos
- O número de bits necessários para codificar um arquivo é dado por

$$B(T) = \sum_{i=1}^{M} f(s) d_T(s)$$

sendo:

- *B(T)* a quantidade de bits necessária para armazenar o arquivo usando a árvore de codificação *T*
- *f(s)* a frequência do símbolo *s*
- d_T o comprimento do código que representa o símbolo *s*
- *M* a quantidade de símbolos da informação

Propriedades da árvore de Huffman

- Cada nó interno tem dois filhos
- As menores frequências estão mais distantes da raiz
- As duas menores frequências são nós irmãos
- O número de bits necessários para codificar um arquivo é dado por

$$B(T) = \sum_{i=1}^{M} f(s) d_T(s)$$

sendo:

- *B(T)* a quantidade de bits necessária árvore de codificação *T*
- *f(s)* a frequência do símbolo *s*
- d_T o comprimento do código que representa o símbolo *s*
- *M* a quantidade de símbolos da informação

A forma como a Árvore de Huffman é construída garante que *B(T)* será a menor possível, considerando um número inteiro de bits por símbolo

Código de Huffman

- No exemplo anterior

Letra	A	I	M	S	Y	/b
Frequência	2	1	3	1	1	2
Código	00	1010	11	1011	100	01

$$B(T) = \sum_{i=1}^{M} f(s) d_T(s)$$

$$B(T) = (2 * 2) + (1 * 4) + (3 * 2) + (1 * 4) + (1 * 3) + (2 * 2) = 25$$

- Número médio de bits por símbolo

$$\frac{B(T)}{M} = \frac{25}{10} = 2,5$$

Construção da árvore de Huffman

- Algoritmo para a construção da árvore de Huffman
  1. Compute a frequência de cada símbolo e armazene em uma tabela de frequências (ou probabilidades)
  2. Para cada símbolo da tabela de frequências
     - Crie uma folha com a frequência associada
     - Insira essa folha (de acordo com a ordenação das frequências) em uma lista ordenada de nós
  3. Enquanto a quantidade de nós na lista ordenada for maior do que 1
     - Remova os dois nós de menor frequência da lista
     - Una-os em um nó pai
     - Faça a frequência do pai ser a soma das frequências dos filhos
     - Insira o nó pai criado na posição correta da lista ordenada de nós

Árvore de Huffman

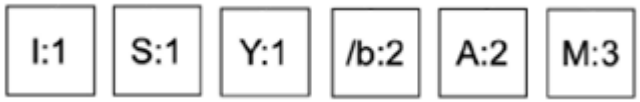
- Exemplo

I		A	M		S	A	M	M	Y

  - Tabela de frequência

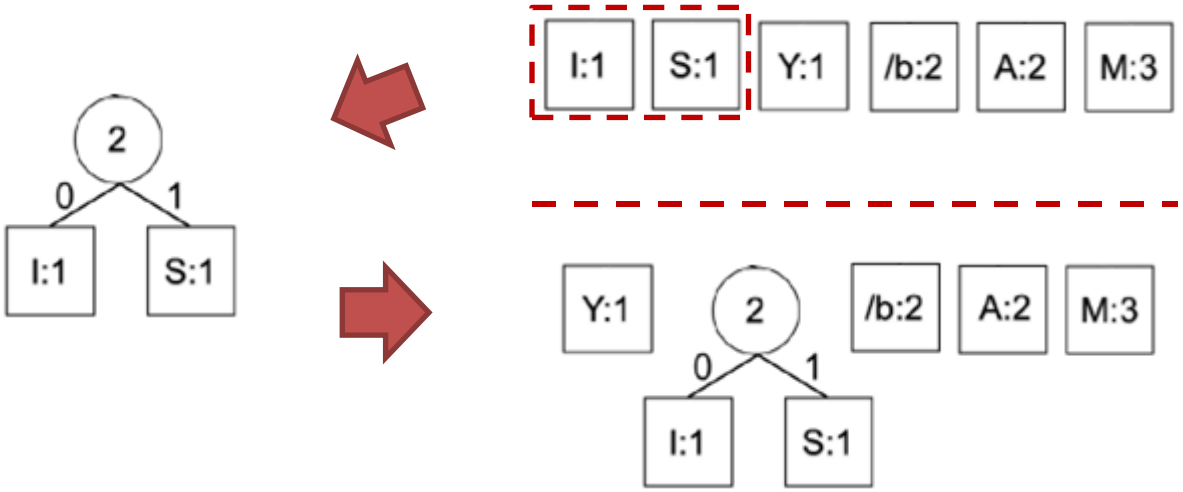
Símbolo	I	S	Y	\b	A	M
Frequência	1	1	1	2	2	3

  - Lista ordenada de nós



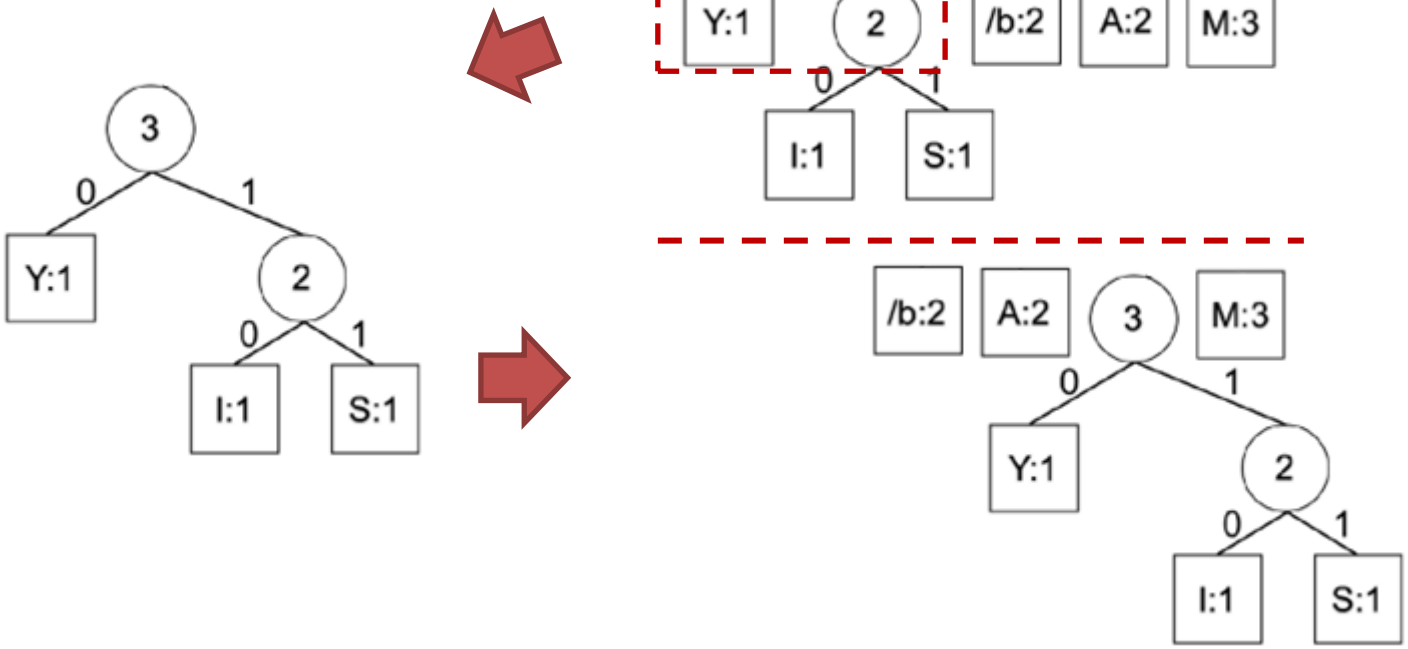
Árvore de Huffman

- Exemplo
  - Lista ordenada de nós



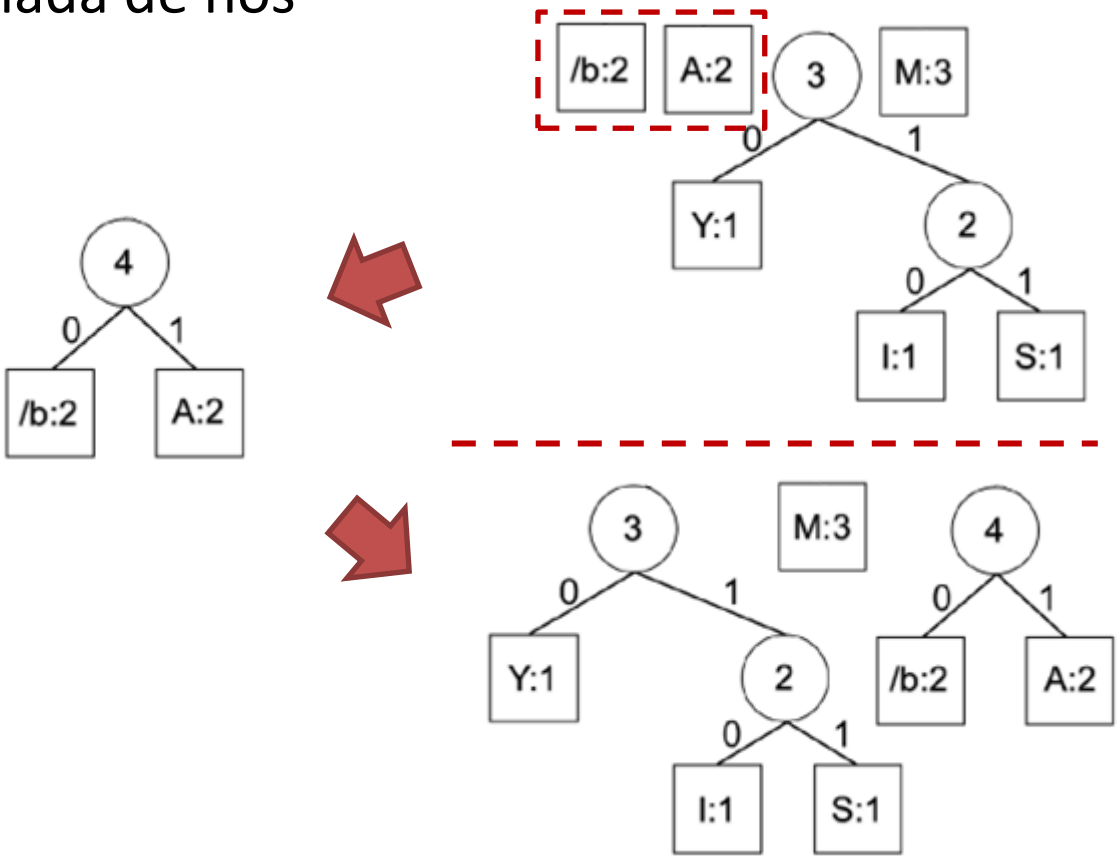
Árvore de Huffman

- Exemplo
  - Lista ordenada de nós



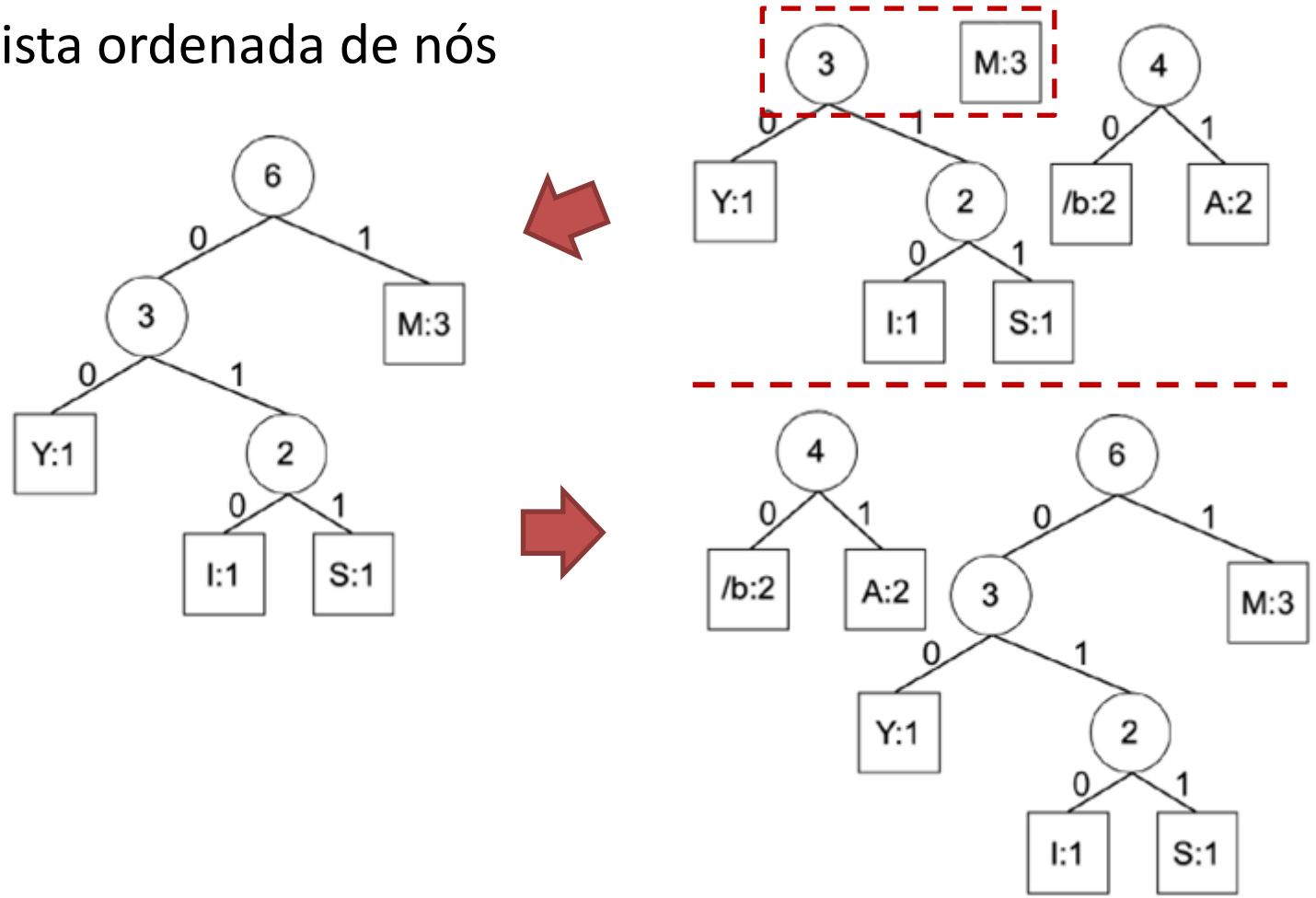
Árvore de Huffman

- Exemplo
  - Lista ordenada de nós



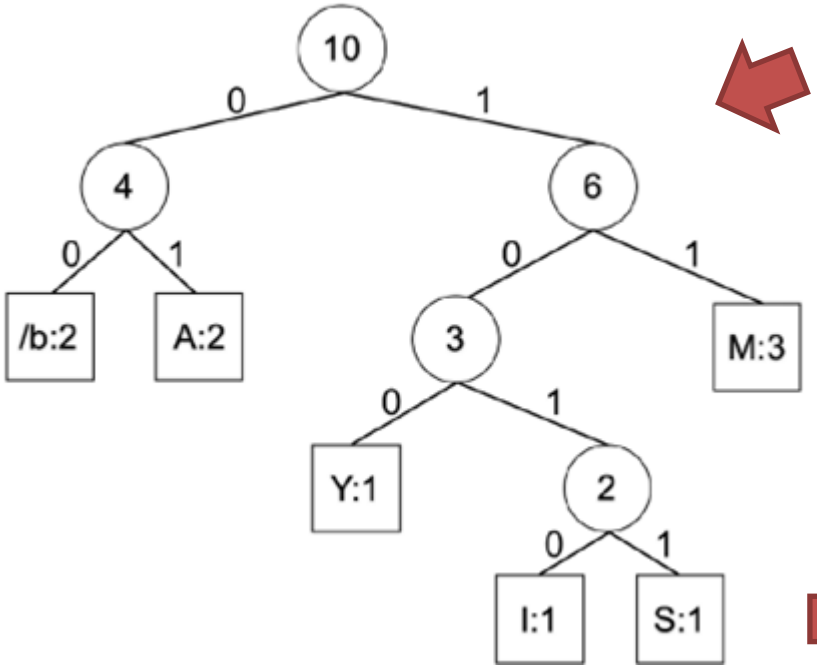
Árvore de Huffman

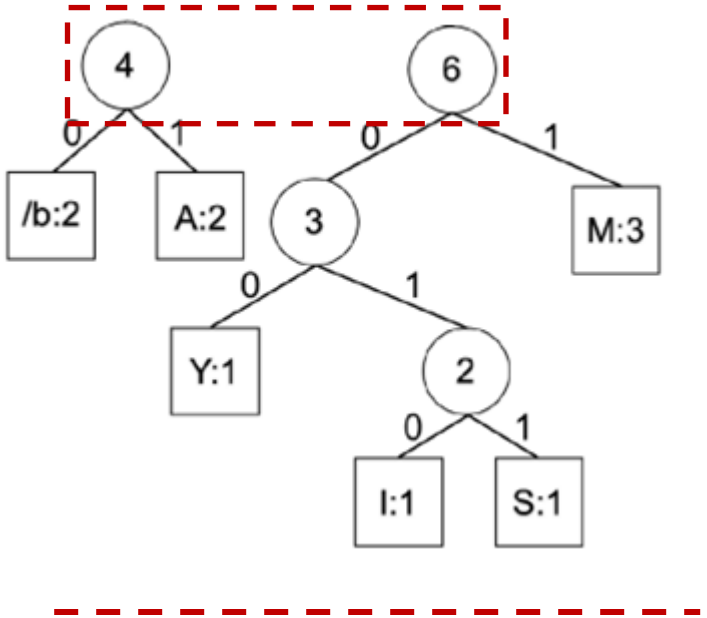
- Exemplo
  - Lista ordenada de nós



Árvore de Huffman

- Exemplo
  - Lista ordenada de nós





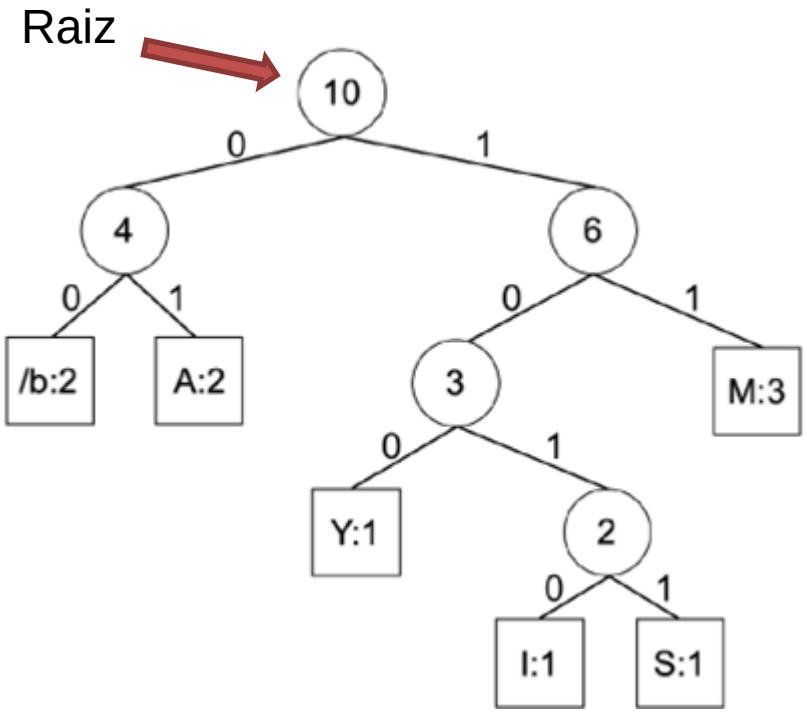
10 → **Apenas um nó, então termina**

Código de Huffman

- Algoritmo para a geração da tabela de códigos
  1. Chame único nó da lista ordenada de nós de raiz da árvore de Huffman
  2. Obtenha o código para cada símbolo capturando a sequência de bits resultante do percurso da raiz até a folha
     - Todos os caminhos da árvore deverão ser percorridos até as folhas
     - Quando um caminho à direita for tomado, concatene o bit 1 ao código sendo gerado
     - Quando um caminho à esquerda for tomado, concatene o bit 0 ao código sendo gerado
     - Quando atingir uma folha, atribua o código gerado ao símbolo na tabela de códigos

Código de Huffman

- Exemplo



Símbolo	Código
\b	00
A	01
M	11
Y	100
I	1010
S	1011

Código de Huffman

- Algoritmo para codificação

  Percorra a árvore e construa a tabela de códigos

  Enquanto o arquivo a ser comprimido ≠ EOF

  1. Para cada byte lido, acesse a tabela de códigos e recupere o código do símbolo
  2. Transfira o código para o *buffer* de saída
     - Some 1 ao contador de bytes codificados
  3. Se o *buffer* de saída encheu, grave-o no arquivo de saída

  Grave a quantidade de bytes codificados no cabeçalho do arquivo de saída

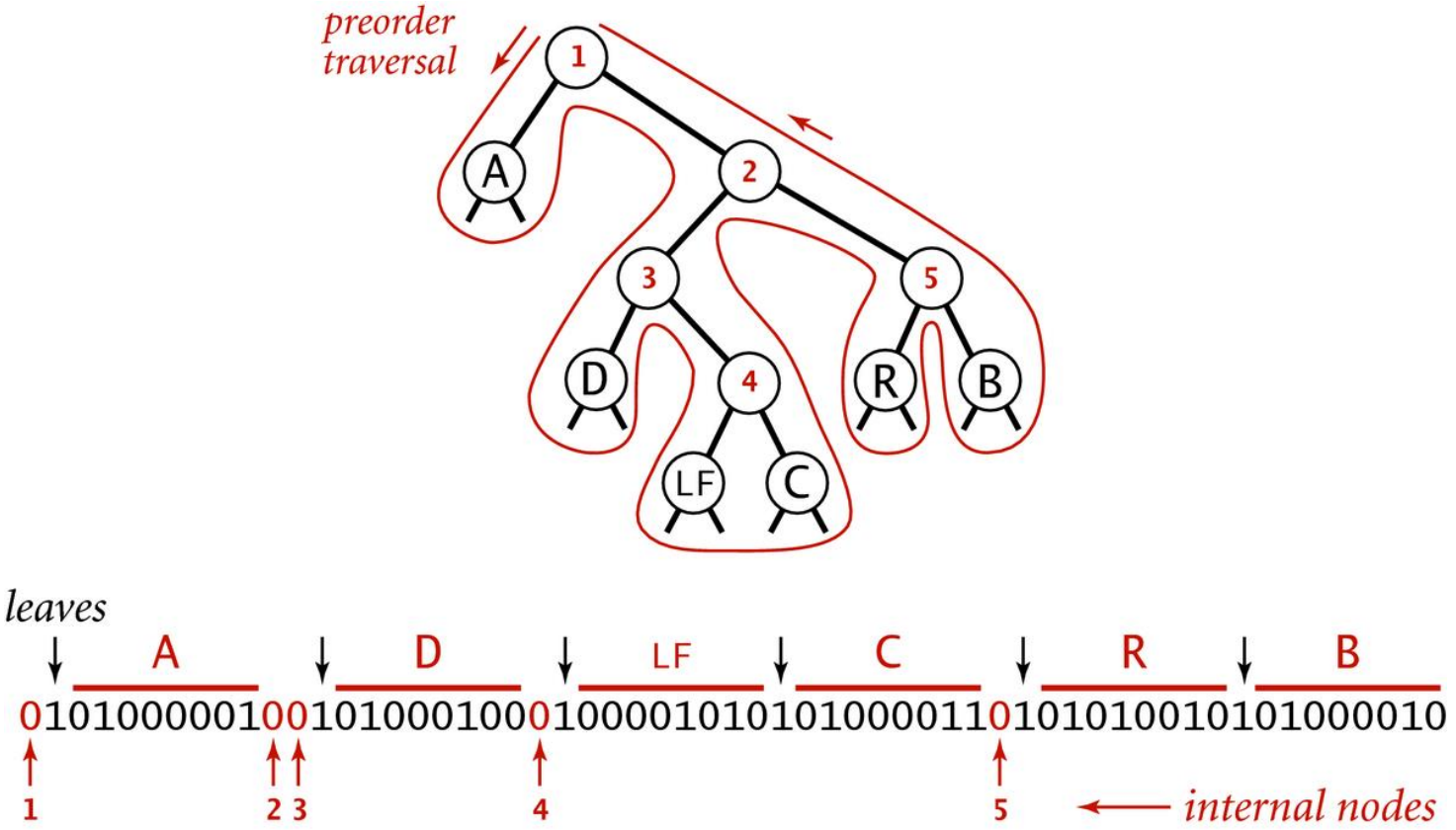
  Acrescente a árvore de Huffman ao arquivo de saída

  Feche os arquivos

Código de Huffman

- Como escrever a árvore de Huffman codificada como uma cadeia de bits?
  - Percorra a árvore em pré-ordem → visite, sistematicamente, a raiz, em seguida a subárvore esquerda e depois a subárvore direita
  - Sempre que visitar um nó interno, escreva um bit 0 na cadeia de saída
  - Sempre que visitar uma folha, escreva um bit 1 seguido pelos 8 bits do caractere associado à folha na cadeia de saída

Código de Huffman



Using preorder traversal to encode a trie as a bitstream

Fonte: Sedgewick e Wayne (2011)

Código de Huffman

- Algoritmo para reconstruir a árvore de Huffman a partir de uma cadeia de bits
- Função reconstruir()
  1. Leia um bit da cadeia de bits
  2. Se o bit for igual a 1
     - Leia um caractere (próximos 8 bits)
     - Crie uma folha referente ao caractere lido e retorne-o (Função CriaNó())
       - Retorne CriaNó (Caractere, NIL, NIL)
  3. Senão
     - Crie um nó interno que terá como filhos esquerdo e direito os retornos de chamadas recursivas à função de reconstruir()
       - Retorne CriaNó (‘\0’, reconstruir(), reconstruir())
       - O ‘\0’ representa o caractere do nó interno, que na verdade não existe

Código de Huffman

- Os comandos *pack* e *unpack* do Unix utilizam código de Huffman
- Para arquivos texto, a codificação pode ser baseada em caracteres ou em palavras
  - Baseada em caracteres: comprime para aprox. 60%
  - Baseada em palavras: comprime entre 25 - 40%
- Não se obtém resultados tão bons para arquivos binários, uma vez que estes tendem a ter uma distribuição mais uniforme de valores

Lempel-Ziv

- É um tipo de compressão baseada em dicionário
- Existem diversas variações dos Códigos Lempel-Ziv
  - Veremos aqui a LZ78
- Os comandos *zip* e *unzip*, e o *compress* e *uncompress* do Unix utilizam codificações Lempel-Ziv
- O dicionário é construído gradualmente a partir da subdivisão da mensagem a ser codificada
  - Na decodificação, um dicionário idêntico é reconstituído a partir da própria codificação
- Diferentemente do código de Huffman, não é necessário adicionar informação "extra" ao arquivo codificado

Codificação Lempel-Ziv

- Usaremos como exemplo uma sequência que utiliza um alfabeto composto de 2 letras (a e b):

  aaababbbaaabaaaaaaabaabb

- Regra para a criação do dicionário
  - Divida a sequência de caracteres em partes tal que cada parte seja a menor sequência de caracteres que ainda não tenha sido vista

    a|aa|b|ab|bb|aaa|ba|aaaa|aab|aabb

Codificação Lempel-Ziv

a|aa|b|ab|bb|aaa|ba|aaaa|aab|aabb

1. Vemos a
2. a já foi visto, agora vemos aa
3. Vemos b
4. a já foi visto, agora vemos ab
5. b já foi visto, agora vemos bb
6. aa já foi visto, agora vemos aaa
7. b já foi visto, agora vemos ba
8. aaa já foi visto, agora vemos aaaa
9. aa já foi visto, agora vemos aab
10. aab já foi visto, agora vemos aabb

Codificação Lempel-Ziv

- Codificação
  - Atribua índices de *1* a *n* para cada parte da sequência
  - Reserve o índice 0 para o caractere nulo

Índice	0	1	2	3	4	5	6	7	8	9	10
string	'\0'	a	aa	b	ab	bb	aaa	ba	aaaa	aab	aabb

  - Uma vez que cada parte da mensagem é a concatenação de uma parte já vista adicionada de um caractere novo, as partes podem ser codificadas por um inteiro (índice do anterior) seguido de um caracter

Índice	1	2	3	4	5	6	7	8	9	10
string	0a	1a	0b	1b	3b	2a	3a	6a	2b	9b

Sequência codificada

Decodificação Lempel-Ziv

- Decodificação
  - Reconstrução do dicionário
  - A decodificação do símbolo atual só depende dos símbolos previamente decodificados (índices menores)
    - Sabemos que o índice zero sempre representa o vazio/nulo

Índice	1	2	3	4	5	6	7	8	9	10
Código	0a	1a	0b	1b	3b	2a	3a	6a	2b	9b
String	a	aa	b	ab	bb	aaa	ba	aaaa	aab	aabb

0a1a0b1b3b2a3a6a2b9b

a aa b ab bb aaa ba aaaa aab aabb

Decodificação Lempel-Ziv

Índice	1	2	3	4	5	6	7	8	9	10
Código	0a	1a	0b	1b	3b	2a	3a	6a	2b	9b
String	a	aa	b	ab	bb	aaa	ba	aaaa	aab	aabb

0a1a0b1b3b2a3a6a2b9b

a

0a1a0b1b3b2a3a6a2b9b

aaa

0a1a0b1b3b2a3a6a2b9b

aaab

0a1a0b1b3b2a3a6a2b9b

aaabab

Decodificação Lempel-Ziv

Índice	1	2	3	4	5	6	7	8	9	10
Código	0a	1a	0b	1b	3b	2a	3a	6a	2b	9b
String	a	aa	b	ab	bb	aaa	ba	aaaa	aab	aabb

0a1a0b1b**3b**2a3a6a2b9b

aaababbb

0a1a0b1b3b**2a**3a6a2b9b

aaababbbaaa

0a1a0b1b3b2a**3a**6a2b9b

aaababbbaaaba

0a1a0b1b3b2a3a**6a**2b9b

aaababbbaaabaaaaa

Decodificação Lempel-Ziv

Índice	1	2	3	4	5	6	7	8	9	10
Código	0a	1a	0b	1b	3b	2a	3a	6a	2b	9b
String	a	aa	b	ab	bb	aaa	ba	aaaa	aab	aabb

0a1a0b1b3b2a3a6a2b9b

aaababbbaaabaaaaaaab

0a1a0b1b3b2a3a6a2b9b

aaababbbaaabaaaaaaabaabb

Lempel-Ziv

- Representação binária da informação codificada
  - **Quantos bits são necessários para representar cada valor inteiro se o índice vai de 0 a *n*?**
    - O maior valor inteiro possível é *n-1*
    - Sendo assim, será preciso no máximo o número de bits necessários para representar o número *n-1*

Índice	1	2	3	4	5	6	7	8	9	10
string	0a	1a	0b	1b	3b	2a	3a	6a	2b	9b

  - Índice 1: nenhum bit (sempre começa com zero)
  - Índice 2: no máximo 1, pois o valor do índice só pode ser 0 ou 1
  - Índice 3-4: no máximo 2, pois o valor do índice fica entre 0 e 3
  - Índice 5-8: no máximo 3, pois o valor do índice fica entre 0 e 7
  - Índice 9-16: no máximo 4, pois o valor do índice fica entre 0 e 15

Lempel-Ziv

- Representação binária da informação codificada
  - Cada símbolo/caracter é representado por 8 bits
  - Cada valor de índice é representado utilizando-se o número de bits necessários para aquela posição

Índice	1	2	3	4	5	6	7	8	9	10
string	0a	1a	0b	1b	3b	2a	3a	6a	2b	9b

a1a00b01b011b010a011a110a0010b1001b

8bits1bit8bits2bits8bits2bits8bits3bits8bits3bits8bits3bits8bits...

Compressão com perda

- Esse tipo de compressão é utilizada quando alguma informação pode ser sacrificada
- É geralmente utilizada com imagem, áudio e vídeo
  - Comprime sem prejudicar o efeito prático
  - Explora as limitações dos sentidos do ser humano
    - Não conseguimos ouvir certas frequências, nem enxergar muitos detalhes (p.e., pequenas variações de cor)
- Exemplos desse tipo de compressão são os formatos JPEG e MPEG