

# SISTEMAS MULTIMÍDIA

## TEXTO 2

Prof.: Danilo Coimbra  
([coimbra.danilo@ufba.br](mailto:coimbra.danilo@ufba.br))



# Compressão em Texto

2

## Método comuns de compressão de texto

- Keyword Encoding
- Técnicas de supressão de sequência repetitivas ou codificação de comprimento de carreira
  - ▣ **Técnica *Run-Length Encoding* (RLE)**
- Técnicas Estatísticas
  - ▣ Codificação de Huffman
  - ▣ Codificação Aritmética
  - ▣ Codificação Shannon-Fano
- Técnica Baseada em dicionário
  - ▣ Lempel-Ziv e Lempel-Ziv-Welsh

# Codificação RLE

3

- Codificação *Run Length Encoding*
  - ▣ ou codificação de comprimento de carreira
- Extensão da técnica de supressão de zeros ou espaços para qualquer tipo de caracter
- Técnica de codificação por entropia
  - ▣ Sem perda
  - ▣ Pode ser utilizada em textos, mas é mais comum nos outros tipos de mídia (imagem, áudio => vídeo)
    - Imagem: PCX, BMP (RLE)

# Codificação RLE

4

- Parte dos dados podem ser compactados por meio de **supressão de sequência de símbolos iguais**
  - Exemplo: AAAAAHHHHHHHHHHHHHHH
- Diferentes notações
  - 5A14H
  - (5,A) (14,H)
  - A,5 H,14 (A,5 é um codeword) (Halsall, 2001)
    - Muito usado para comprimir valores binários
      - 0000011111110000111111111100
      - 0,5 1,7 0,4 1,10 0,2
  - !5A!14H

# Codificação RLE

5

- Sequências idênticas são substituídas por um símbolo especial (“!”), número de ocorrências (**n**) e o(s) símbolo(s) repetidos (“**c**”)

$!<n><c>$

- Exemplo: UHHHHHHIMMG12223

- U!6HIMMG12223

- Taxa de compressão:  $13 / 16 = \sim 0.813$

- $1 - 0.813 = \sim 0.19$ , ou  $\sim 19\%$  de taxa de compressão

# Codificação RLE

6

- Eficiência da compressão depende do dado de entrada
- Técnica não é utilizada para sequências menores que 4
  - Por quê?
- Caso o símbolo especial ocorra no dado de entrada, ele deve/pode ser substituído por dois símbolos
  - Entrada: U!HIIIIID
  - Saída: U!!H!5ID

# Codificação RLE

7

- O algoritmo pode ser facilmente otimizado, substituindo sequências maiores que um byte
  - ▣ Como ?
  
- Utilizando um caractere especial de fim
  - ▣ Entrada: UFYUGDUFHUFHUFHUFHUFHBFD
  
  - ▣ Saída: UFYUGD!**5UFH**\$BFD

# Codificação RLE

8

- É vantajoso aplicar a RLE quando houver grandes agrupamentos de símbolos iguais
  - ▣ Diminuir a redundância
  
- E se não ocorrer muita redundância?
  - ▣ REELEMENT
    - 1R 2E 1L 1E 1M 1E 1N 1T
  
  - ▣ Taxa
    - 16:9- > ~ 1.78
    - Expansão, compressão negativa

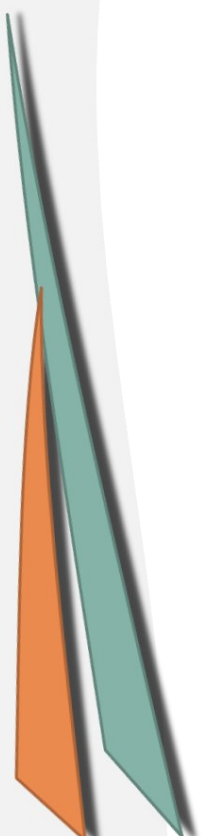


# Codificação RLE

9

Além do texto..

- As principais aplicações são em imagens
  - ▣ Binárias
  - ▣ Coloridas: imagens com grandes espaços envolvendo uma só cor



# Compressão em Texto

10

## Métodos comuns de compressão de texto

- Keyword Encoding
- Técnicas de supressão de sequência repetitivas ou codificação de comprimento de carreira
  - ▣ Técnicas de supressão de zeros ou espaços
  - ▣ Técnica *Run-Length Encoding* (RLE)
- **Técnicas Estatísticas**
  - ▣ **Codificação de Huffman**
  - ▣ Codificação Aritmética
  - ▣ Codificação Shannon-Fano
- Técnica Baseada em dicionário
  - ▣ Lempel-Ziv e Lempel-Ziv-Welsh

# Codificação de Huffman

11

- Codificação estatística
  - ▣ Compressão simétrica
    - Codificador e decodificador possuem complexidade idêntica
  
- Principal característica
  - ▣ Atribuir **menos** bits a símbolos que aparecem com **maior** frequência, e
  - ▣ **mais** bits para símbolos que aparecem com **menor** frequência

# Codificação de Huffman

12

## Exemplo

- Suponha um arquivo de 1.000 caracteres: e t x z
- Quantos bits são necessários para representar cada um dos caracteres?
  - ▣ 2 bits para cada um dos 4 símbolos  
 $e=00$  ;  $t=01$  ;  $x=10$  ;  $z=11$
  - ▣ Tamanho do arquivo:
    - 1 caractere = 1 byte --> 1000 bytes = 8000 bits
    - $2 * 8000 = 16.000$  bits

# Codificação de Huffman

13

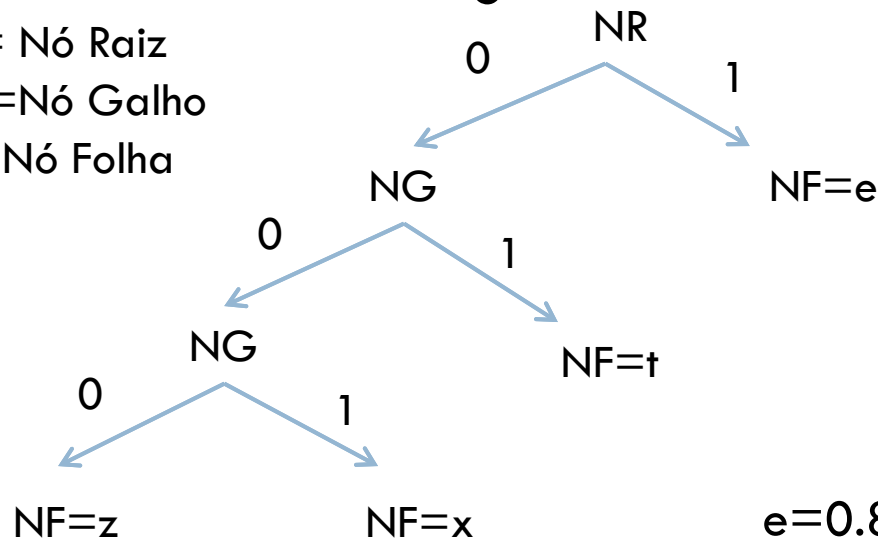
- ❑ E se considerarmos a codificação de Huffman?
  - ❑ Usando quantidade de bits diferentes para representar os símbolos de acordo com suas probabilidades?
    - Mais bits : menor frequência
    - Menos bits: maior frequência
  - ❑ Probabilidades  
 $e=0.8$  ;  $t= 0.16$  ;  $x= 0.02$  ;  $z= 0.02$
  - ❑ Bits  
 $e= 1$  ;  $t = 01$  ;  $x= 001$  ;  $z= 000$
  - ❑ Tamanho do arquivo:  
 $8.000(1*0.8+2*0.16+3*0,02+3*0,02) = \mathbf{9.920 \text{ bits}}$
  - ❑ Apesar de x e z terem mais bits, o número será menor pois eles ocorrem menos vezes

# Codificação de Huffman

14

- A codificação envolve a criação de uma árvore binária não balanceada (Árvore de Huffman)
  - ▣ Os caracteres estão nas folhas
  - ▣ Aresta à direita de um nó recebe valor 1; e a aresta à esquerda recebe valor 0
  - ▣ Percorrendo-se a árvore da raiz em direção às folhas obtém-se os códigos de cada caractere

NR= Nó Raiz  
NG=Nó Galho  
NF=Nó Folha



$e=0.8$  ;  $t= 0.16$  ;  $x= 0.02$  ;  $z= 0.02$

# Codificação de Huffman

15

## Passos para construir a árvore de Huffman

1) Ordenar os símbolos por ordem decrescente de probabilidade

Símbolos	Probabilidade/ frequência
e	0.8
t	0.16
x	0.02
z	0.02

2) Contrair os 2 símbolos de menor probabilidade em um símbolo hipotético cuja probabilidade é a soma destes

e	0.8
t	0.16
x	0.02
z	0.02



e	0.8
t	0.16
(x,z)	0.04

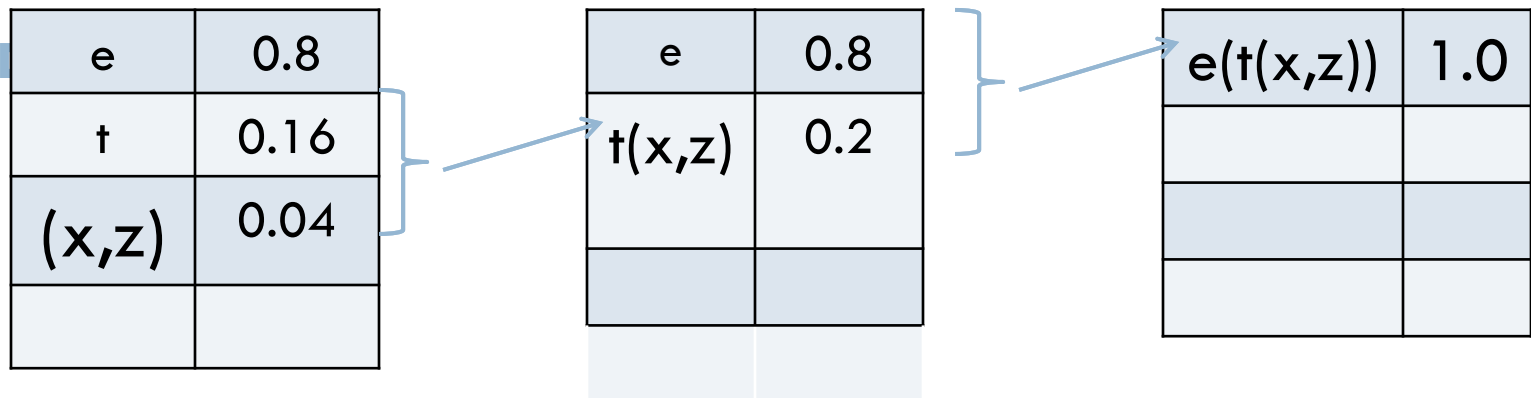
# Codificação de Huffman

16

## Passos para construir a árvore de Huffman

3) Repetir os passos 1 e 2 anteriores até que todos os símbolos estejam agregados num símbolo hipotético com probabilidade 1

■ Empates são resolvidos aleatoriamente





# Codificação de Huffman

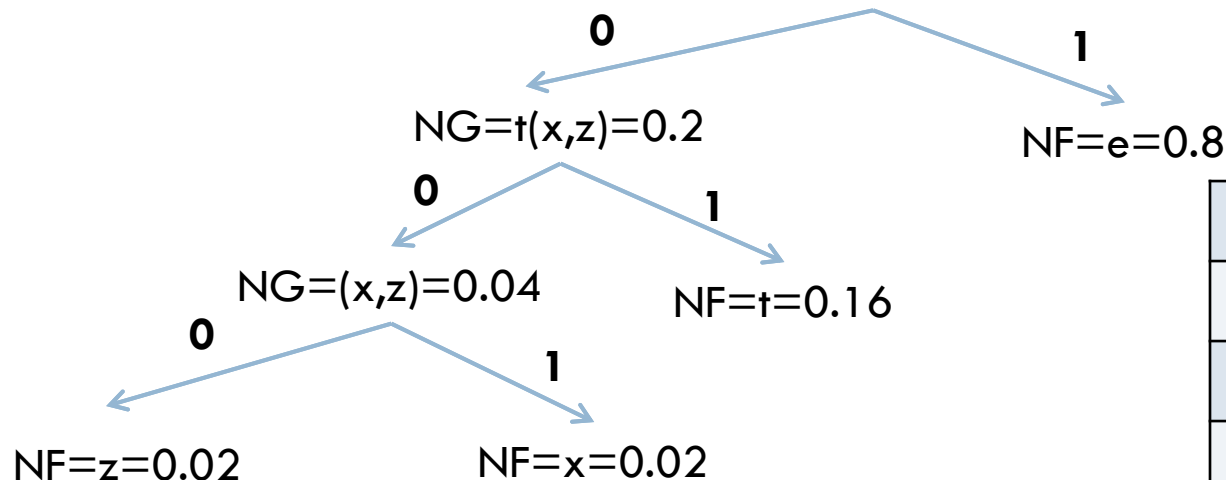
17

## 4) Construir a árvore com base nas tabelas

- Na ordem inversa da construção das tabelas
- aresta à direita de um nó = 1; e a aresta à esquerda = 0

e	0.8	e	0.8	e	0.8	$e(t(x,z))$	1.0
t	0.16	t	0.16	$t(x,z)$	0.2		
x	0.02	$(x,z)$	0.04				
z	0.02						

$$NR = e(t(x,z)) = 1.0$$



e	1
t	01
x	001
z	000

# Codificação de Huffman

18

- Árvore ótima (de Huffman):
  - ▣ Basta verificar, da esquerda para a direita, e de baixo para cima, se os pesos estão em ordem crescente
- Árvore de Huffman tem a propriedade do prefixo
  - ▣ Nenhum código é prefixo de outro código

# Codificação de Huffman

19

## □ Codificação:

- Basta substituir os caracteres pelos respectivos códigos

- Tabela de códigos

e	1
t	01
x	001
z	000

- No exemplo, a string eeettxz será codificada como:

- 1110101001000

# Codificação de Huffman

20

## □ Decodificação:

- Basta usar o código de Huffman como índice para percorrer a árvore
- No exemplo, o primeiro bit do código 11110101001000 é 1. **A partir da raiz da árvore,** percorre-se a mesma à direita (1). Se encontrou um nó folha, escreve o caracter correspondente, volta para raiz e pega próximo bit do código. Senão, pega próximo bit do código.

# Codificação de Huffman

21

- Diferentes árvores binárias válidas para mesmo dado de entrada
  - ▣ Quando probabilidades são iguais
- Operação computacional mais custosa
  - ▣ Adição de floats (probabilidades)
- No decodificador
  - ▣ Realiza uma simples verificação na tabela de Huffman
  - ▣ Tabela de Huffman é parte do fluxo de dados ou é conhecida pelo decodificador

# Codificação de Huffman

22

## □ Observações:

- Ambos, codificador e decodificador devem conhecer a tabela (ou árvore) de códigos.
- Se a tabela é enviada/codificada junto com os dados, ocorre sobrecarga de dados (*overhead*)
- O decodificador pode conhecer a tabela com antecedência
  - Ex.: Análise estatística do uso dos caracteres em uma determinada língua
  - Esse método não é exato
    - Alguns textos não vão atingir o máximo de compressão que poderiam

# Compressão em Texto

23

## Métodos comuns de compressão de texto

- Keyword Encoding
- Técnicas de supressão de sequência repetitivas ou codificação de comprimento de carreira
  - ▣ Técnicas de supressão de zeros ou espaços
  - ▣ Técnica *Run-Length Encoding* (RLE)
- **Técnicas Estatísticas**
  - ▣ Codificação de Huffman
  - ▣ **Codificação Aritmética**
  - ▣ Codificação Shannon-Fano
- Técnica Baseada em dicionário
  - ▣ Lempel-Ziv e Lempel-Ziv-Welsh

# Codificação Aritmética

24

- ❑ Codificação estatística
  - ❑ Entropia
- ❑ Patenteada pela **IBM**
- ❑ Não gera um novo código para cada símbolo
  - ❑ Como acontece com Huffman
- ❑ Atribui-se um código a cada conjunto de dados
  - ❑ Número real



# Codificação Aritmética

25

- Método de Huffman atinge o valor da Entropia apenas em algumas situações
  - ▣ Depende da probabilidade de aparecimento dos caracteres no texto
- Codificação Aritmética atinge valores mais próximos da Entropia
  - ▣ Mais complexa que Huffman
  - ▣ Iremos estudar apenas o modo básico

# Codificação Aritmética

26

- String a ser codificada: **went.**
- Probabilidades:
  - ▣  $e = 0,3; n = 0,3; t = 0,2; w = 0,1; . = 0,1$
  - ▣  $.$  = terminador de string
- Conjunto de caracteres deve ser dividido no intervalo de 0 a 1, respeitando-se a proporção das probabilidades

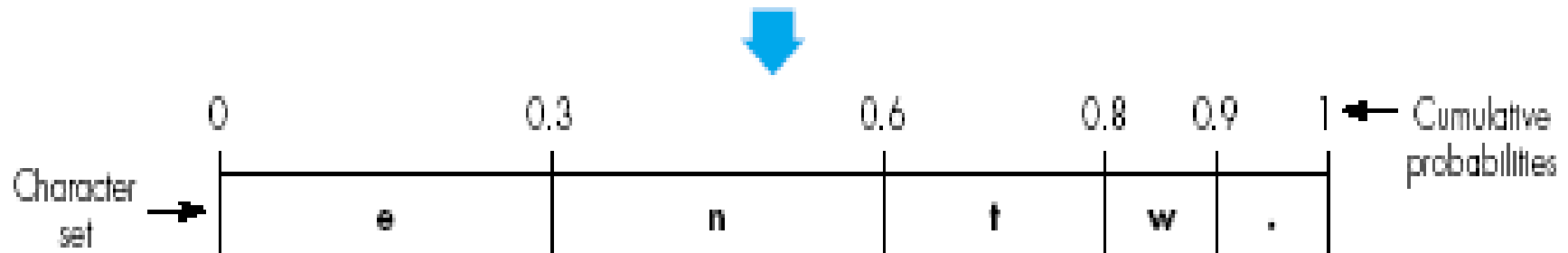
# Codificação Aritmética

27

- Cada subintervalo, na ordem da mensagem, é subdividido respeitando-se as proporções

Example character set and their probabilities:

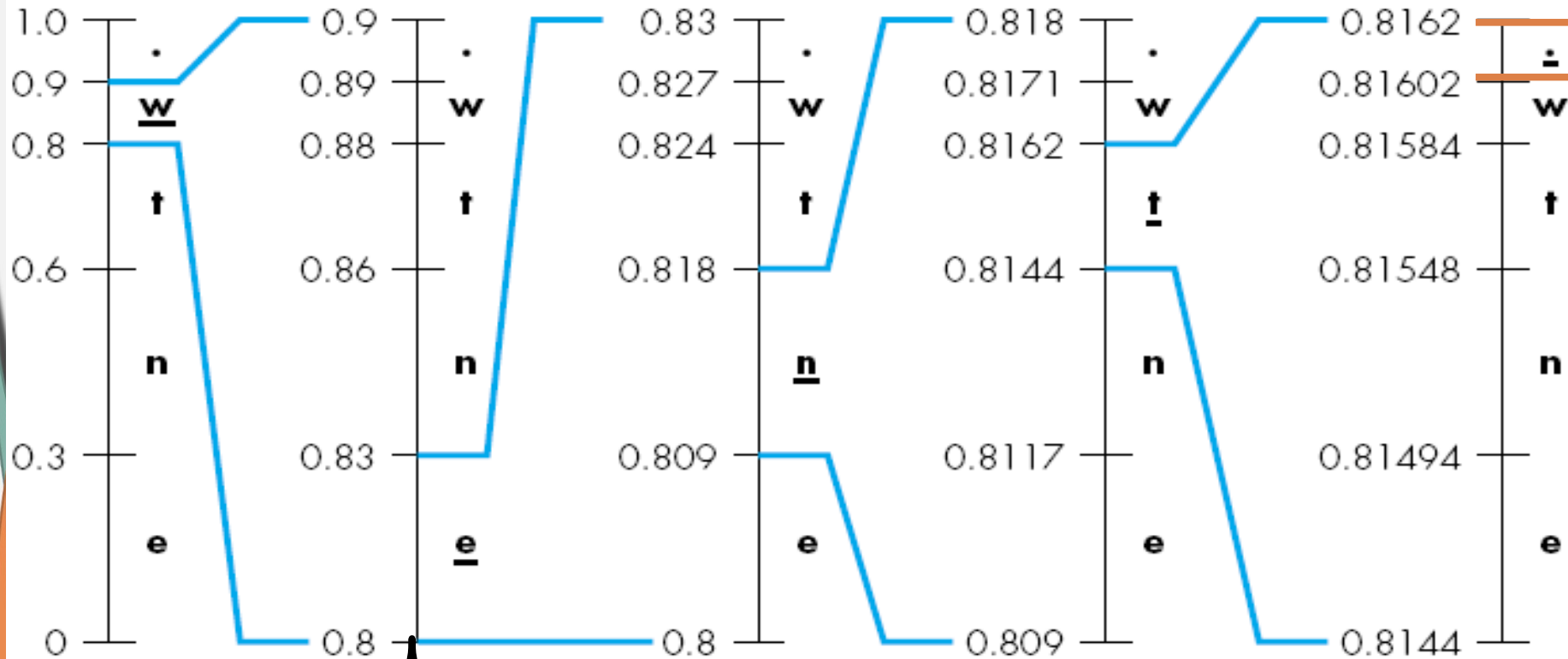
$$e = 0.3, n = 0.3, t = 0.2, w = 0.1, . = 0.1$$



# Codificação Aritmética

went.

28



$$\begin{aligned}
 e &= 0.8 + (0.3 * (0.9 - 0.8)) = 0.83 \\
 n &= 0.83 + (0.3 * (0.9 - 0.8)) = 0.86 \\
 t &= 0.86 + (0.2 * (0.9 - 0.8)) = 0.88 \\
 w &= 0.88 + (0.1 * (0.9 - 0.8)) = 0.89 \\
 . &= 0.89 + (0.1 * (0.9 - 0.8)) = 0.9
 \end{aligned}$$

$$e = 0,3; n = 0,3; t = 0,2; w = 0,1; . = 0,1$$

Calculando o segundo intervalo  
segundo as proporções acima

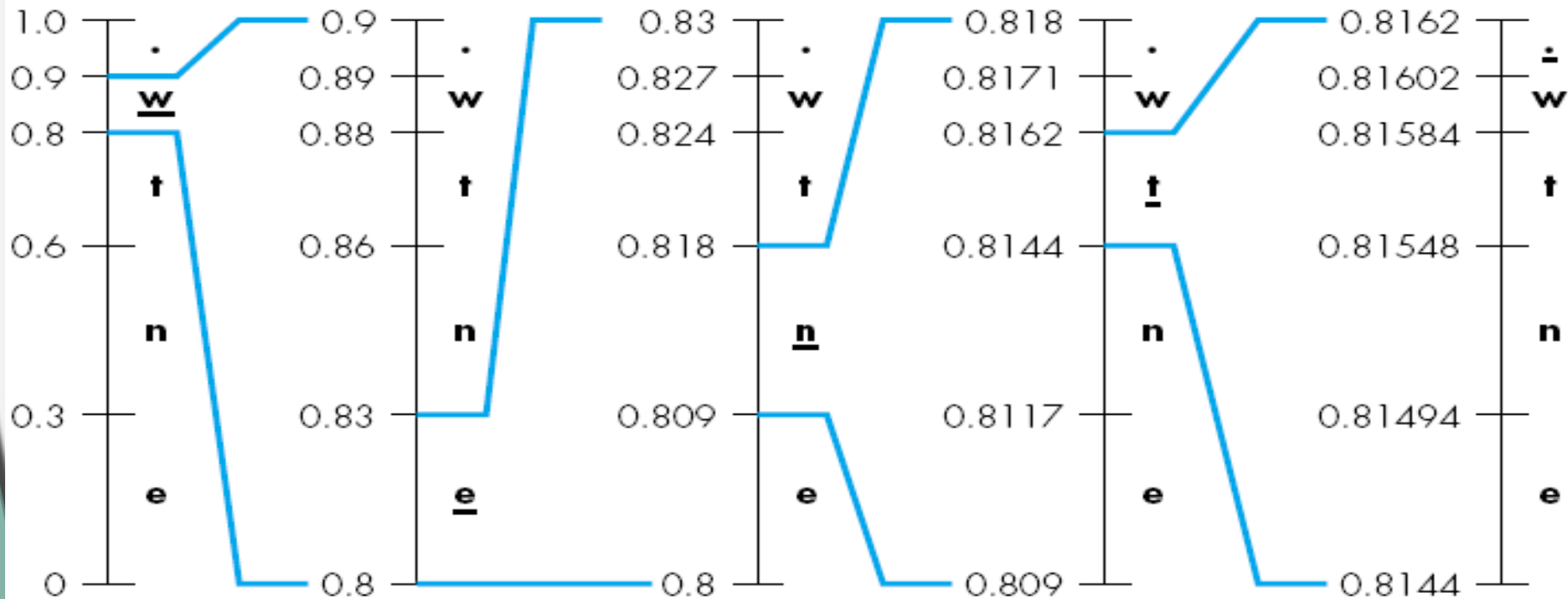
# Codificação Aritmética

29

- Nesse exemplo, o código pode ser qualquer número entre 0,81602 e 0,8162
  - ▣ 0,8161, por exemplo
  
- Decodificador conhece o alfabeto, as probabilidades e os intervalos
  - ▣ Então pode seguir o mesmo processo do codificador para decodificar a mensagem 0,8161

# Codificação Aritmética

30



0,8161  $\Rightarrow$  primeiro caracter é “w”, pois 0,8161 está no intervalo 0,8-0,9.

O segundo é “e”, pois 0,8161 está no intervalo 0,8-0,83.

E assim por diante.

# Codificação Aritmética

31

- Nesse método, o número de dígitos no código cresce linearmente de acordo com o tamanho da string
  - ▣ Quanto maior a string, maior o número
- Logo, o número **máximo** de caracteres em uma string é determinado pela precisão de ponto flutuante na máquina destino
  - ▣ Strings grandes podem ser quebradas em duas ou mais substrings

# Codificação Aritmética

32

- Se o alfabeto for grande
  - ▣ A probabilidade máxima é baixa e portanto o código de Huffman comporta-se melhor que a codificação aritmética
- Quanto maior for o tamanho da sequência de dados mais se aproxima do valor da entropia
- É preciso somente estimar a probabilidade do alfabeto de entrada
  - ▣ Não há necessidade de preservar a árvore como na codificação Huffman



# Codificação Shannon-Fano

33

## Características:

- ❑ Codificação estatística
- ❑ Algoritmo sem perda
- ❑ Simétrico

# Codificação Shannon-Fano

34

A codificação de Shannon-Fano constrói a árvore de codificação do seguinte modo

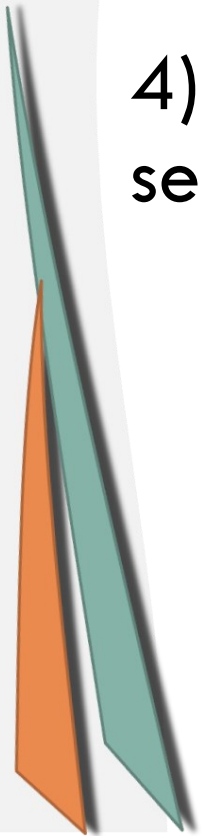
- 1) Ordene os símbolos de acordo com suas frequências/probabilidades
- 2) Divida recursivamente em dois grupos, cada uma com aproximadamente o mesmo número de contagem (soma das frequências)
  - ▣ Um grupo recebe o valor 0 e o outro 1

# Codificação Shannon-Fano

35

3) Procedimento 2 é repetido até ficar um símbolo em cada grupo

4) O código para cada símbolo é formado pela sequência resultante de valores 0 e 1



# Codificação Shannon-Fano

36

## Exemplo

Símbolos	Probabilidade
C	0.20
F	0.10
A	0.30
D	0.10
E	0.10
B	0.20

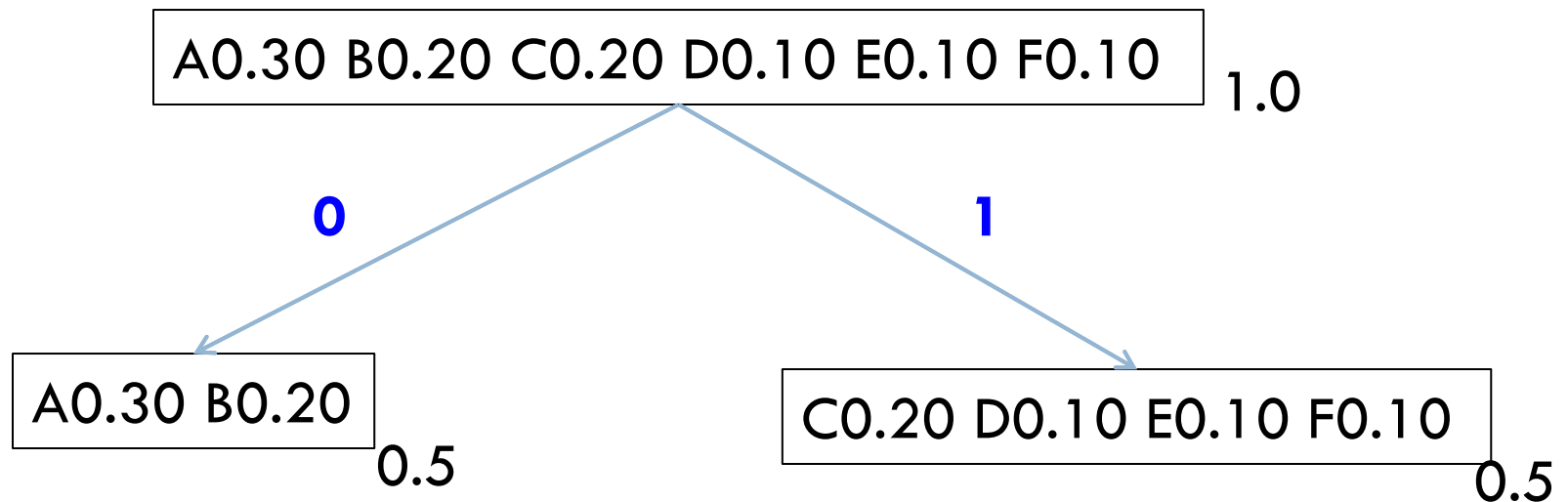
## Passo 1

Símbolos	Probabilidade
A	0.30
B	0.20
C	0.20
D	0.10
E	0.10
F	0.10

# Codificação Shannon-Fano

37

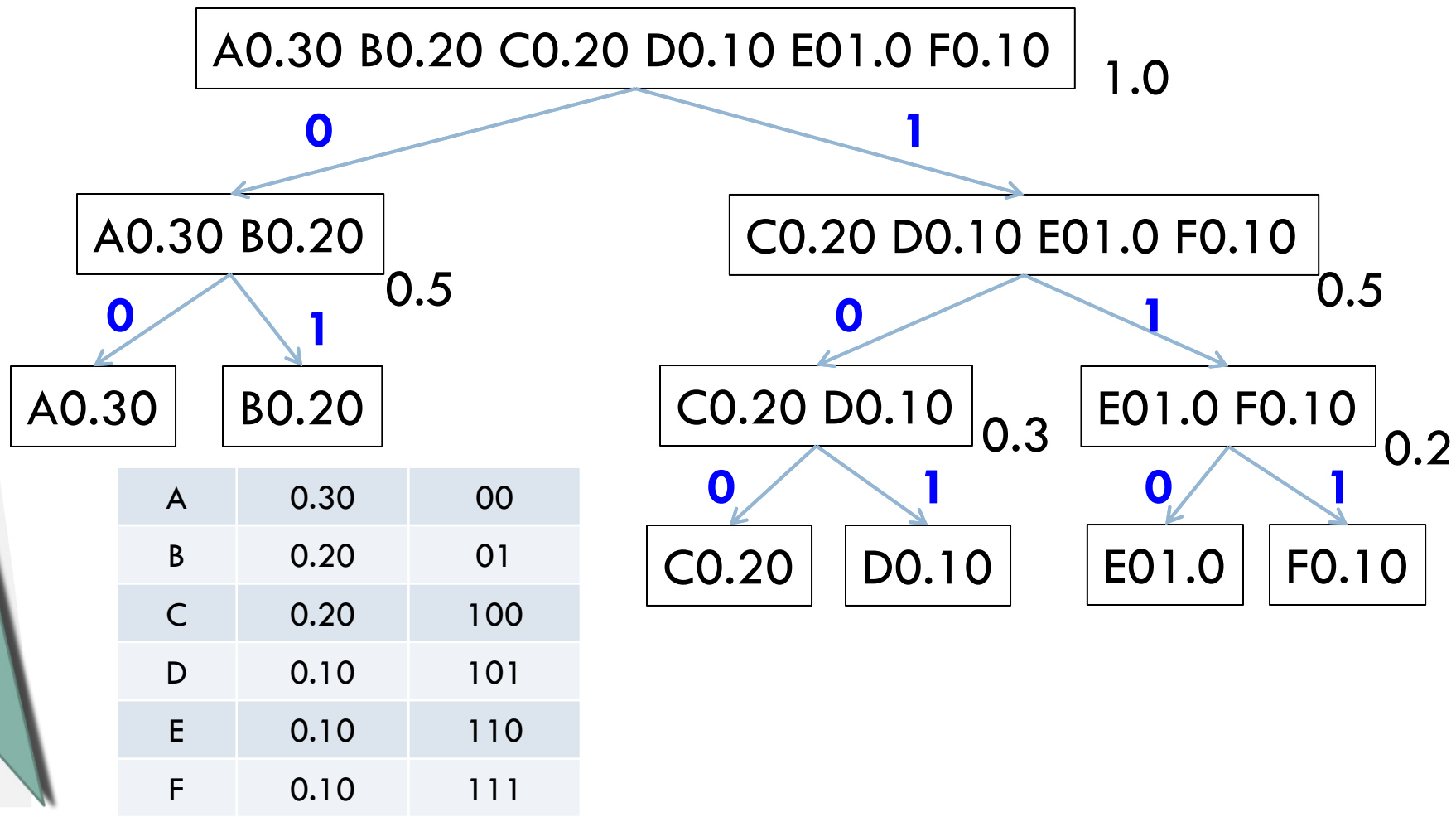
- Passo 2: divisão em 2 grupos atribuindo 1 e 0



# Codificação Shannon-Fano

38

- Passo 4: gerar o código de cada símbolo



# Compressão em Texto

39

## Método comuns de compressão de texto

- Keyword Encoding
- Técnicas de supressão de sequência repetitivas ou codificação de comprimento de carreira
  - ▣ Técnicas de supressão de zeros ou espaços
  - ▣ **Técnica *Run-Length Encoding* (RLE)**
- Técnicas Estatísticas
  - ▣ Codificação de Huffman
  - ▣ Codificação Aritmética
  - ▣ Codificação Shannon-Fano
- Técnica Baseada em dicionário
  - ▣ Lempel-Ziv e Lempel-Ziv-Welsh

# Codificação Lempel-Ziv-Welch (LZW)

40

- Nome do algoritmo é derivado dos nomes de seus autores
  - ▣ Abraham Lempel, Jakob Ziv, Terry Welch
- É um algoritmo
  - ▣ Simétrico
  - ▣ Sem perdas
  - ▣ Adaptativo
- Utiliza a estratégia de compressão baseada em dicionário



# Codificação Lempel-Ziv-Welch

41

- Baseada no algoritmo Lempel-Ziv
- Técnica baseada em dicionário
  - ▣ Em vez de codificar caracteres, codifica strings, as quais são armazenadas em uma tabela (dicionário)
- O codificador e o decodificador constroem o dicionário **dinamicamente**

# Codificação Lempel-Ziv-Welch (LZW)

42

## Algoritmo

$S$  = primeiro símbolo do fluxo de dados de entrada;

Enquanto (existir dados de entrada){

$c$  = próximo símbolo do fluxo de dados de entrada ( $S_i$ );

    Se  $(S + c)$  existir no dicionário

$S = S + c$ ;

    Senão{

        Saída = código de  $S$ ;

        Adicionar  $(S + c)$  ao dicionário, criando um novo código;

$S = c$ ;

    }


}

Saída = código de  $S$  (último símbolo)

# Codificação Lempel-Ziv-Welch

43

- O algoritmo anterior visa preencher essa tabela

S	$c$	Saída (Ficheiro comprimido)	Código	Sequência
Símbolos dos dados de entrada ( $S_i$ )	Próximo símbolo dos dados de entrada ( $c_i$ )	Código de S	Código correspondente à sequência	Símbolo ou conjunto de símbolos que formam o dicionário
				
		<b>Código comprimido</b>		
		<b>Dicionário</b>		



# Codificação Lempel-Ziv-Welch

44

- Exemplo de funcionamento utilizando a cadeia **ABACABA**
- O primeiro passo consiste em inicializar a tabela de códigos com todos os caracteres existentes na string que pretendemos comprimir:
- Tabela de Códigos

S	c	Saída	Código	Sequência
-	-	-	1	A
-	-	-	2	B
-	-	-	3	C

# Codificação Lempel-Ziv-Welch

45

Exemplo de funcionamento utilizando a cadeia **ABACABA**

- 2º passo: leitura da sequência símbolo a símbolo e início do preenchimento

S	c	Saída	Código	Sequência
-	-	-	1	A
-	-	-	2	B
-	-	-	3	C
A	B	1	4	AB

S=A

c=B

AB não existe no dicionário

Saída=1 (código de A)

Adiciona AB ao dicionário, com código 4

S=B (c)

## Algoritmo

```
S = primeiro símbolo do fluxo de dados de entrada;  
Enquanto (existir dados de entrada){  
  c = próximo símbolo do fluxo de dados de entrada (Si);  
  Se (S + c) existir no dicionário  
    S = S + c;  
  Senão{  
    Saída = código de S;  
    Adicionar (S + c) ao dicionário, criando um novo código;  
    S = c;  
  }  
}  
Saída = código de S (último símbolo)
```

# Codificação Lempel-Ziv-Welch

46

Exemplo de funcionamento utilizando a cadeia **ABACABA**

S	c	Saída	Código	Sequência
-	-	-	1	A
-	-	-	2	B
-	-	-	3	C
A	B	1	4	AB
B	A	2	5	BA

S=B

c=A

BA não existe no dicionário

Saída=2 (código de B)

Adiciona BA ao dicionário, com código 5

S=A (c)

## Algoritmo

```
S = primeiro símbolo do fluxo de dados de entrada;  
Enquanto (existir dados de entrada){  
  c = próximo símbolo do fluxo de dados de entrada (Si);  
  Se (S + c) existir no dicionário  
    S = S + c;  
  Senão{  
    Saída = código de S;  
    Adicionar (S + c) ao dicionário, criando um novo código;  
    S = c;  
  }  
}  
Saída = código de S (último símbolo)
```

# Codificação Lempel-Ziv-Welch

47

Exemplo de funcionamento utilizando a cadeia **ABACABA**

S	c	Saída	Código	Sequência
-	-	-	1	A
-	-	-	2	B
-	-	-	3	C
A	B	1	4	AB
B	A	2	5	BA
A	C	1	6	AC

S=A

c=C

AC não existe no dicionário

Saída=1 (código de A)

Adiciona AC ao dicionário, com código 6

S=C (c)

## Algoritmo

```
S = primeiro símbolo do fluxo de dados de entrada;  
Enquanto (existir dados de entrada){  
  c = próximo símbolo do fluxo de dados de entrada (Si);  
  Se (S + c) existir no dicionário  
    S = S + c;  
  Senão{  
    Saída = código de S;  
    Adicionar (S + c) ao dicionário, criando um novo código;  
    S = c;  
  }  
}  
Saída = código de S (último símbolo)
```

# Codificação Lempel-Ziv-Welch

48

Exemplo de funcionamento utilizando a cadeia **ABACABA**

S	c	Saída	Código	Sequência
-	-	-	1	A
-	-	-	2	B
-	-	-	3	C
A	B	1	4	AB
B	A	2	5	BA
A	C	1	6	AC
C	A	3	7	CA

S=C

c=A

CA não existe no dicionário

Saída=3 (código de C)

Adiciona CA ao dicionário, com código 7

S=A (c)

## Algoritmo

```
S = primeiro símbolo do fluxo de dados de entrada;  
Enquanto (existir dados de entrada){  
  c = próximo símbolo do fluxo de dados de entrada (Si);  
  Se (S + c) existir no dicionário  
    S = S + c;  
  Senão{  
    Saída = código de S;  
    Adicionar (S + c) ao dicionário, criando um novo código;  
    S = c;  
  }  
}  
Saída = código de S (último símbolo)
```



# Codificação Lempel-Ziv-Welch

49

**ABACABA**

S	c	Saída	Código	Sequência
-	-	-	1	A
-	-	-	2	B
-	-	-	3	C
A	B	1	4	AB
B	A	2	5	BA
A	C	1	6	AC
C	A	3	7	CA
A	B	-	-	-

S=A

c=B

AB existe no dicionário

S=AB

## Algoritmo

```
S = primeiro símbolo do fluxo de dados de entrada;  
Enquanto (existir dados de entrada){  
  c = próximo símbolo do fluxo de dados de entrada (Si);  
  Se (S + c) existir no dicionário  
    S = S + c;  
  Senão{  
    Saída = código de S;  
    Adicionar (S + c) ao dicionário, criando um novo código;  
    S = c;  
  }  
}  
Saída = código de S (último símbolo)
```

# Codificação Lempel-Ziv-Welch

50

**ABACABA**

S	c	Saída	Código	Sequência
-	-	-	1	A
-	-	-	2	B
-	-	-	3	C
A	B	1	4	AB
B	A	2	5	BA
A	C	1	6	AC
C	A	3	7	CA
A	B	-	-	-
AB	A	4	8	ABA

S=AB

c=A

ABA não existe no dicionário

Saída=4

Adiciona ABA, com código 8

S=A (c)

## Algoritmo

```
S = primeiro símbolo do fluxo de dados de entrada;  
Enquanto (existir dados de entrada){  
  c = próximo símbolo do fluxo de dados de entrada (Si);  
  Se (S + c) existir no dicionário  
    S = S + c;  
  Senão{  
    Saída = código de S;  
    Adicionar (S + c) ao dicionário, criando um novo código;  
    S = c;  
  }  
}  
Saída = código de S (último símbolo)
```

# Codificação Lempel-Ziv-Welch

51

**ABACABA**

S	c	Saída	Código	Sequência
-	-	-	1	A
-	-	-	2	B
-	-	-	3	C
A	B	1	4	AB
B	A	2	5	BA
A	C	1	6	AC
C	A	3	7	CA
A	B	-	-	-
AB	A	4	8	ABA
A	-	1	-	-

S=A

c=-

Saída = 1 (código de A (S))

Obtemos a seguinte sequência codificada:  
**121341**

# Codificação Lempel-Ziv-Welch

52

Além do texto..

- Imagem

- É utilizado nos formatos GIF e TIFF

- O maior problema de implementação do algoritmo envolve o gerenciamento da tabela que implementa o dicionário

- Interessante utilizar tabela Hashing para facilitar a busca

# Referências

53

- ❑ Halsall, F. Multimedia Communications: Applications, Networks, Protocols, and Standards, Addison-Wesley Publishing, 2001. ISBN: 0201398184. Capítulos 2 e 4.
- ❑ Mandal, M. K. Multimedia Signals and Systems. Kluwer Academic Publishers, 2002. ISBN: 1402072708. Capítulo 6
- ❑ Ribeiro, Nuno, and José Torres. "Tecnologias de Compressão Multimédia." (2009).
- ❑ Notas de aula da equipe de professores de natureza da informação. Universidade Federal do ABC.