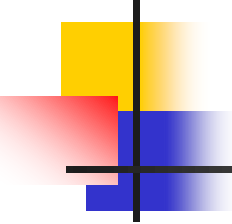




# Compressão de Dados

---

## Estruturas de Dados II



# Compressão de Dados *vs.* Compactação de Dados

---

- Processos distintos:
  - Compressão: reduz a quantidade de bits para representar um dado
    - Dado: imagem, texto, vídeo, áudio
  - Compactação: união de dados que não estejam unidos
    - [Compressão sem perdas]



# Compressão de Dados

---

- A compressão destina-se também a eliminação de redundância
  - Realizada através de uma regra, chamada de código ou protocolo, que, quando seguida, elimina os bits redundantes de informações, de modo a diminuir seu tamanho nos arquivos.



# Omissão de sequências repetidas (RLE)

---

- Considere uma imagem (8-bits) que foi processada de forma que apenas os objetos com um brilho acima de um certo valor foram identificados e mantidos, e todos os demais foram setados para uma cor de fundo representada por um valor de pixel igual a 0 (digamos, preto).
- Esta imagem é um tipo de matriz esparsa, e matrizes como estas são boas candidatas ao uso de uma técnica de compressão denominada *run-length encoding (RLE)*

# Omissão de sequências repetidas (RLE)



- Foto galáxia NGC 891. O espaço vazio nesta imagem astronômica é representando por sequências repetidas e é um bom candidato para compressão.



# Omissão de sequências repetidas (RLE)

---

- Algoritmo
  - Leia a sequência copiando os valores para um arquivo destino sequencialmente, exceto quando o mesmo valor ocorre mais de uma vez sucessivamente.
  - Quando dois ou mais pixels iguais ocorrem em sequência, substitua-os pelos seguintes 3 bytes:
    - o byte especial que indica run-length
    - o valor que se repete
    - o número de vezes que o valor se repete (no máximo 256. Porque?)



# Omissão de sequências repetidas (RLE)

---

- Para a sequência
  - 22 23 24 24 24 24 24 24 24 25 26 26 26  
26 26 26 25 24
- Usando 0xff como código de *run-length*
  - 22 23 ff 24 07 25 ff 26 06 25 24
- Garante redução de espaço sempre?



# Compressão de Dados

---

- Além de eliminar a redundância, os dados são comprimidos para:
  - Economizar espaço em dispositivos de armazenamento, como discos rígidos
  - Ganhar desempenho (diminuir tempo) em transmissões (transmissão mais rápida)
  - Processamento sequencial mais rápido





# Classificação

---

- Com Perdas x Sem Perdas
- Simétricos x Assimétricos
- Adaptativo x Não Adaptativo
- Quanto a Operação



# Com Perdas x Sem Perdas

---

- Forma mais conhecida
- Sem perdas: os dados obtidos após a descompressão são idênticos aos dados que se tinha antes da compressão
- Com perdas: os dados obtidos após a descompressão não são idênticos aos originais
  - Ex.: vídeo e áudio: usa as limitações do ser humano que não ouve altas frequências ou não vê muitos detalhes em imagens



# Simétricos x Assimétricos

---

- Diferenças de complexidade entre a compressão e a descompressão
- Simétricos: quando a compressão e a descompressão são realizadas executando-se métodos ou algoritmos idênticos ou bem semelhantes
  - Exemplo: LZW



# Simétricos x Assimétricos

---

- Assimétricos: quando o método de compressão é mais complexo que o de descompressão (ou em casos raros, o de descompressão é mais complexo que o de compressão)
  - Útil quando se irá comprimir apenas uma vez, mas descomprimir várias vezes (músicas em formato MP3 são um bom exemplo disso).
  - Exemplo: LZ77



# Adaptativos x Não Adaptativos

---

- Não adaptativos: regras não variam de acordo com os dados, nem a medida que os dados são lidos
  - Huffman com leitura prévia dos dados (frequência)
- Adaptativos
  - Se adaptam aos dados a medida que estes são processados
    - Exemplo: LZ77 e LZ78 (inviável criar dicionários padronizados)



# Quanto a Operação

---

- métodos estatísticos ou métodos de aproximação de entropia
  - usam as probabilidades de ocorrência dos símbolos no fluxo de dados e alteram a representação de cada símbolo ou grupo de símbolos
  - visam reduzir o número de bits usados para representar cada símbolo ou grupo de símbolos
  - Exemplo: Código de Huffman



# Quanto a Operação

---

- métodos baseados em dicionários ou métodos de redução de redundância
  - Usam dicionários ou outras estruturas similares de forma a eliminar repetições de símbolos (frases) redundantes ou repetidas
    - Exemplos: LZ77 e LZ78



# Código de Huffman

---





# Código de Huffman

---

- Usados em muitos algoritmos de compressão (gzip (.gz), bzip2, etc.)
- Algoritmo para a **compressão de arquivos**, principalmente arquivos textos
- Atribui **códigos menores** para símbolos mais frequentes e **códigos maiores** para símbolos menos frequentes
- **Código** é um conjunto de bits



# Código de Huffman

---

- Suponha um arquivo de dados com 100 caracteres que se deseja armazenar de forma compacta
- Existem muitas formas de representar esse arquivo
- Uma delas seria através da construção de um **código de caractere binário**, onde cada caractere é representado por uma única string binária



# Código de Huffman

---

- Se usarmos um código de tamanho fixo, 3 bits são necessários para representar 6 caracteres
- Esse método requer 300 bits para codificar todo o arquivo

abc = 000.001.010

	a	b	c	d	e	f
Frequência	45	13	12	16	9	5
Código de tamanho fixo	000	001	010	011	100	101
Código de tamanho variável	0	101	100	111	1101	1100



# Código de Huffman

- Um código de tamanho variável tem um desempenho melhor, uma vez que fornece códigos menores aos caracteres mais frequentes e maiores para os menos frequentes

- Requer no caso exemplo:

$$(45*1) + (13*3) + (12*3) + (16*3) + (9*4) + (5*4) = 224$$

abc = 0.101.100

	a	b	c	d	e	f
Frequência	45	13	12	16	9	5
Código de tamanho fixo	000	001	010	011	100	101
Código de tamanho variável	0	101	100	111	1101	1100



# Código de Huffman

## Códigos de Prefixo

---

- Um código de prefixo é um código de comprimento variável no qual nenhuma palavra de código é prefixo de outra palavra de código
  - Exemplo  
 $a = 0, b = 110, c = 111, d = 10$



# Código de Huffman

## Códigos de Prefixo

---

- Ambiguidade
  - Exemplo  
ACBA = 01010

<u>Símbolo</u>	<u>Huffman</u>
A	0
B	01
C	1

Os dois bits em vermelho são (A e C) ou (B)?

Código de A é prefixo do código de B



# Código de Huffman

## Códigos de Prefixo

---

- Códigos de prefixo são desejáveis porque eles simplificam a decodificação (não há ambiguidade)
  - O processo de decodificação necessita de uma representação de código de prefixo de forma a facilitar o processo
    - Árvore binária

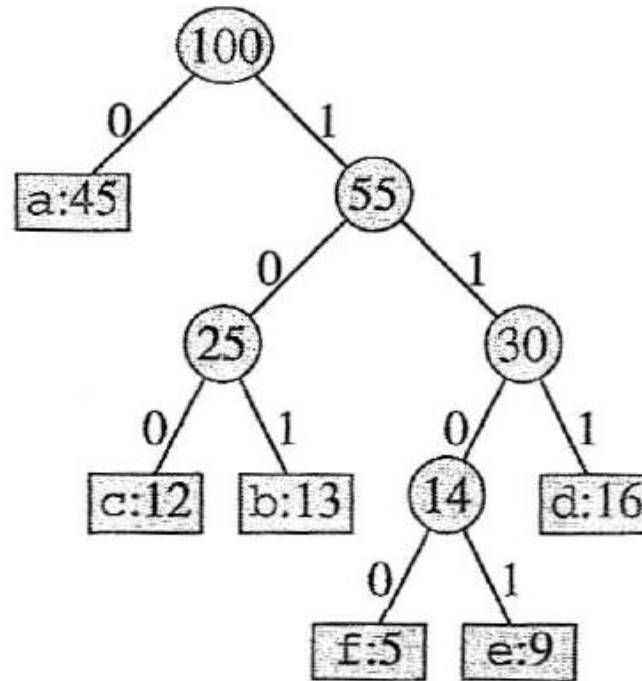
001011101 = 0.0.101.1101 = aabe

	a	b	c	d	e	f
Código de tamanho variável	0	101	100	111	1101	1100

# Código de Huffman

## Códigos de Prefixo

0 = filho a esquerda  
1 = filho a direita  
nós folhas = caracteres



001011101 = 0.0.101.1101 = aabe

	a	b	c	d	e	f
Código de tamanho variável	0	101	100	111	1101	1100





# Construção do Código de Huffman

---

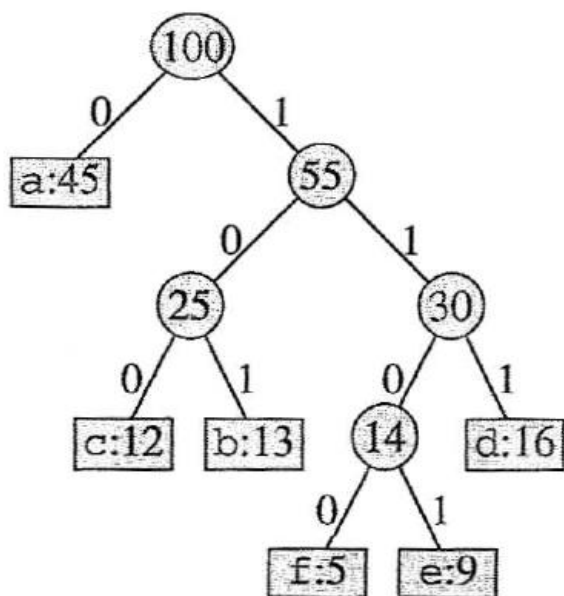
- Dada uma tabela de frequências, como determinar o melhor conjunto de códigos, ou seja, o conjunto que comprimirá mais os símbolos?
- Huffman desenvolveu um algoritmo para isso
  - Dado: tabela de frequências dos  $N$  símbolos de um alfabeto
  - Objetivo: atribuir códigos aos símbolos de modo que os mais frequentes tenham códigos menores (menos bits)

# Construção do Código de Huffman

- Idéia básica

- Construir uma árvore binária tal que:

- (a) suas folhas sejam os N símbolos do alfabeto
- (b) cada ramo da árvore seja um valor 1 (direita) ou 0 (esquerda)
  - Isso é uma convenção, o contrário também funciona
- O código de um símbolo será a sequência de bits dos ramos da raiz até sua posição na árvore



# Construção do Código de Huffman

## Algoritmo

```
Huffman (C)
n  $\leftarrow$  |C|
Q  $\leftarrow$  C
for i  $\leftarrow$  1 to (n-1) do
    allocate a new node z
    left[z]  $\leftarrow$  x  $\leftarrow$  Extract-Min(Q)
    right[z]  $\leftarrow$  y  $\leftarrow$  Extract-Min(Q)
    f[z]  $\leftarrow$  f[x] + f[y]
    Insert(Q,z)
return Extract-Min(Q) // Return the root of the tree
```

C = conjunto de n caracteres com suas respectivas frequências  
f[c] = frequência do caractere c  
Q = fila de prioridade mínima

# Construção do Código de Huffman

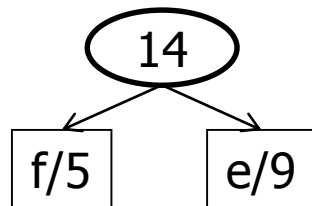
Algoritmo

```
Huffman (C)
n ← |C|
Q ← C
for i ← 1 to (n-1) do
    allocate a new node z
    left[z] ← x ← Extract-Min(Q)
    right[z] ← y ← Extract-Min(Q)
    f[z] ← f[x] + f[y]
    Insert(Q,z)
return Extract-Min(Q) // Return the root of the tree
```

n = 6

Q = f/5, e/9, c/12, b/13, d/16, a/45

i = 1



x = f/5

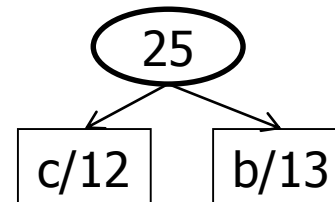
y = e/9

f[z] = 5 + 9 = 14

Q = c/12, b/13, 14/14, d/16, a/45

i = 2

Q = c/12, b/13, 14/14, d/16, a/45



x = c/12

y = b/13

f[z] = 12 + 13 = 25

Q = 14/14, d/16, 25/25, a/45

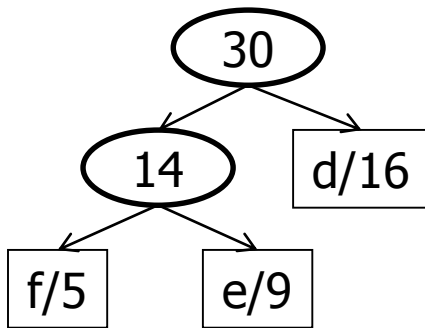
# Construção do Código de Huffman

Algoritmo

```
Huffman (C)
n ← |C|
Q ← C
for i ← 1 to (n-1) do
    allocate a new node z
    left[z] ← x ← Extract-Min(Q)
    right[z] ← y ← Extract-Min(Q)
    f[z] ← f[x] + f[y]
    Insert(Q,z)
return Extract-Min(Q) // Return the root of the tree
```

i = 3

Q = 14/14, d/16, 25/25, a/45



x = 14/14

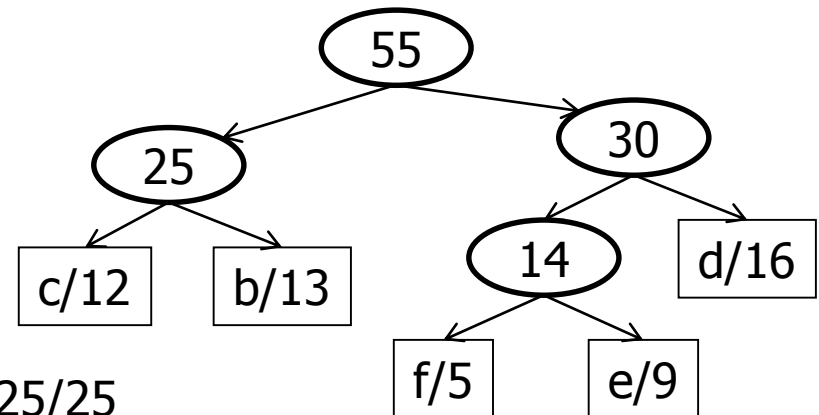
y = d/16

f[z] = 14 + 16 = 30

Q = 25/25, 30/30, a/45

i = 4

Q = 25/25, 30/30, a/45



x = 25/25

y = 30/30

f[z] = 25 + 30 = 55

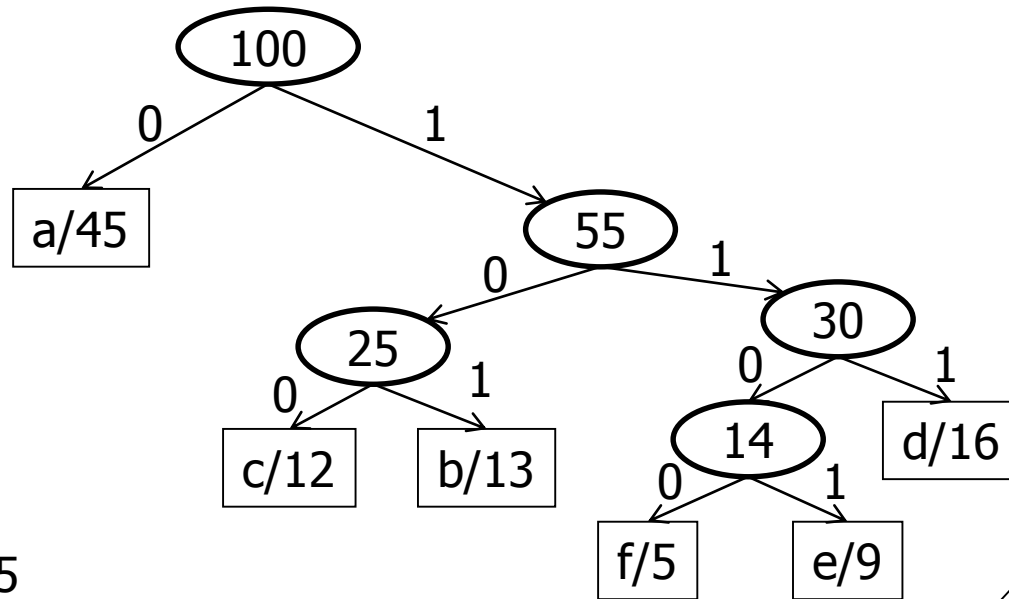
Q = a/45, 55/55

# Construção do Código de Huffman

Algoritmo

```
Huffman (C)
n ← |C|
Q ← C
for i ← 1 to (n-1) do
  allocate a new node z
  left[z] ← x ← Extract-Min(Q)
  right[z] ← y ← Extract-Min(Q)
  f[z] ← f[x] + f[y]
  Insert(Q,z)
return Extract-Min(Q) // Return the root of the tree
```

i = 5  
Q = a/45, 55/55



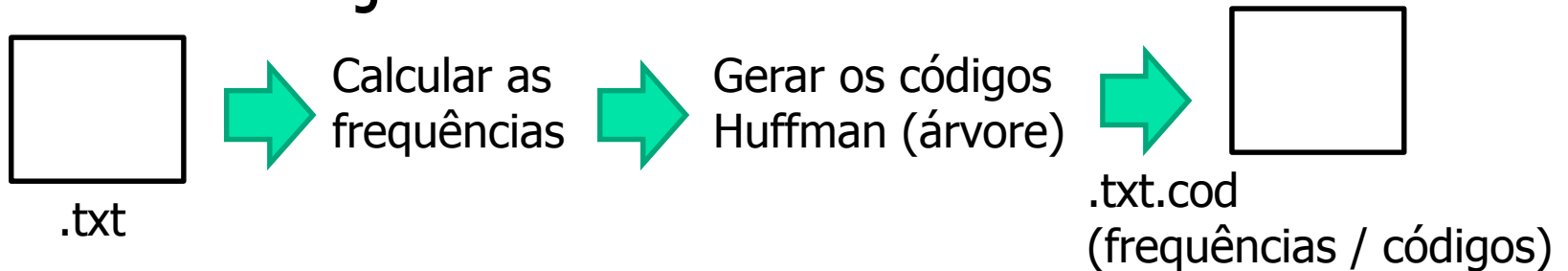
x = a/45  
y = 55/55  
f[z] = 45 + 55 = 100  
Q = 100/100

i = 6  
Q = ∅  
Aponta para 100/100

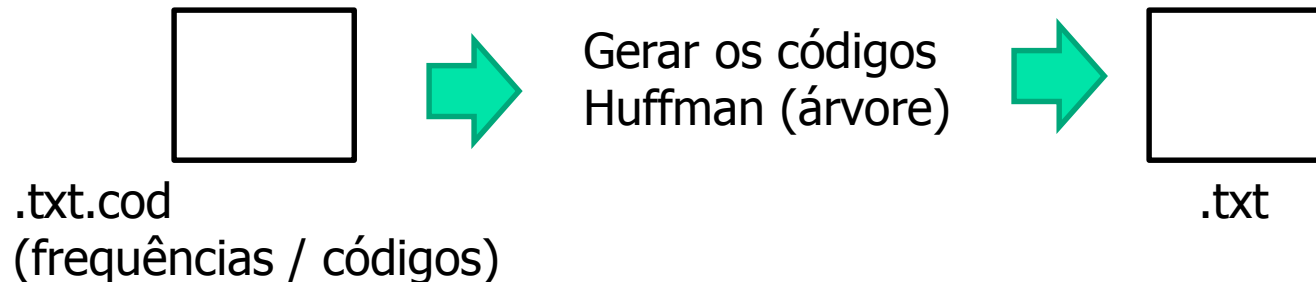
# Código de Huffman

## Processo de Compressão

### Codificação



### Decodificação



# Código de Huffman

## Processo de Compressão

- Comentários sobre a implementação

Operadores bit a bit no C  
Operam sobre **char** e **int**

&	AND
	OR
^	OR exclusivo (XOR)
>>	Deslocamento à direita ( $/2^N$ )
<<	Deslocamento à esquerda ( $*2^N$ )

X=7	0000 0111	7
X<<1	0000 1110	14
X<<3	0111 0000	112
X>>1	0011 1000	56
X>>2	0000 1110	14



# Exercício

## Código de Huffman

Código de Bits

S	a	b	d	e	f	h	i	k	n	o	r	s	t	u	v
9	5	1	3	7	3	1	1	1	4	1	5	1	2	1	1

Quantos bits são necessários para codificar a sequência?

*Em caso de empate, considerar ordem alfabética, e minúscula antes de maiúscula.*

S	
r	
a	
e	
n	
d	
f	
t	
u	
v	
b	
h	
i	
k	
o	
s	



# Codificação de Shannon-Fano

---



# Codificação de Shannon-Fano

---

- É um método que gera códigos de tamanho variável para cada símbolo do conjunto de dados de acordo com sua probabilidade de ocorrência
- É anterior ao de Huffman, e apesar de bastante eficiente e prático, gera resultados sub-ótimos
  - É possível provar que o código gerado pelo método de Huffman gera códigos livre de prefixo de tamanho ótimo para cada símbolo



# Codificação de Shannon-Fano

---

- O algoritmo usado no PKZIP e outros compactadores de arquivos em formato ZIP usa a codificação de Shannon-Fano em conjunto com um algoritmo de janela deslizante baseado em LZ77



# Codificação de Shannon-Fano

---

- [1] Contabilize as probabilidades de ocorrência de cada símbolo
  - Para efeitos práticos, a contagem do número de ocorrências de cada símbolo é o suficiente
- [2] Ordene a lista de probabilidades em ordem decrescente
- [3] Separe a lista em duas partes de forma que cada uma dessas partes tenha aproximadamente a mesma probabilidade
  - A soma das probabilidades de cada símbolo de uma parte seja o mais próximo possível de 50%



# Codificação de Shannon-Fano

---

[4] A cada uma dessas partes atribua o primeiro dígito como sendo 0 (primeira parte) ou 1 (segunda parte)

[5] Volte ao passo [3] até que todas as folhas contenham apenas um símbolo

**(Cont.)**



# Codificação de Shannon-Fano

## Exemplo

---

- Comprimir:  
AAAAAABBBBBBCCCCDDDEEF

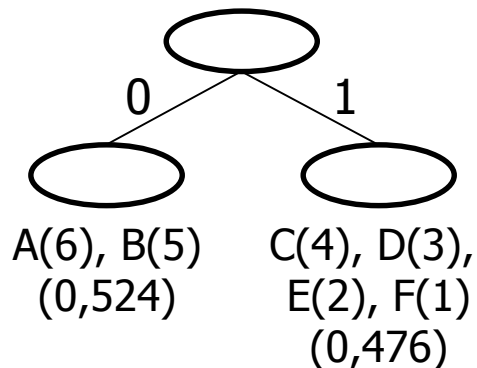
Símbolo	A	B	C	D	E	F
Frequência	6	5	4	3	2	1

# Codificação de Shannon-Fano

## Exemplo

$$6+5+4+3+2+1 = 21$$

Símbolo	A	B	C	D	E	F
Frequência	6	5	4	3	2	1
Probabilidade	0,286	0,238	0,190	0,143	0,095	0,048



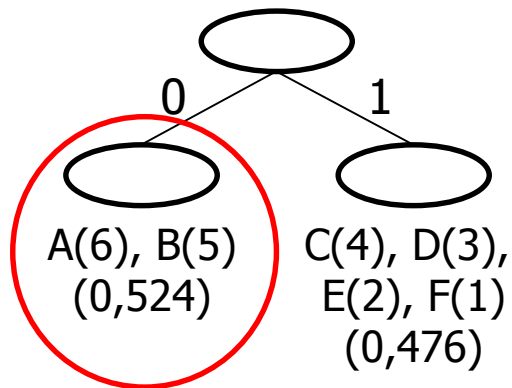


# Codificação de Shannon-Fano

## Exemplo

$$6+5+4+3+2+1 = 21$$

Símbolo	A	B	C	D	E	F
Frequência	6	5	4	3	2	1
Probabilidade	0,286	0,238	0,190	0,143	0,095	0,048



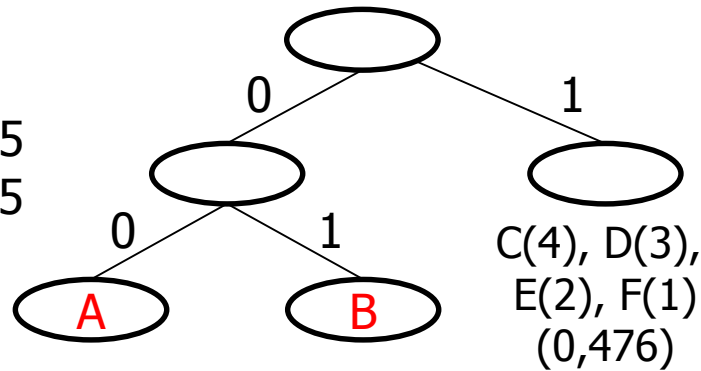
# Codificação de Shannon-Fano

## Exemplo

$$6+5 = 11$$

$$6(A) = 0,545$$

$$5(B) = 0,455$$



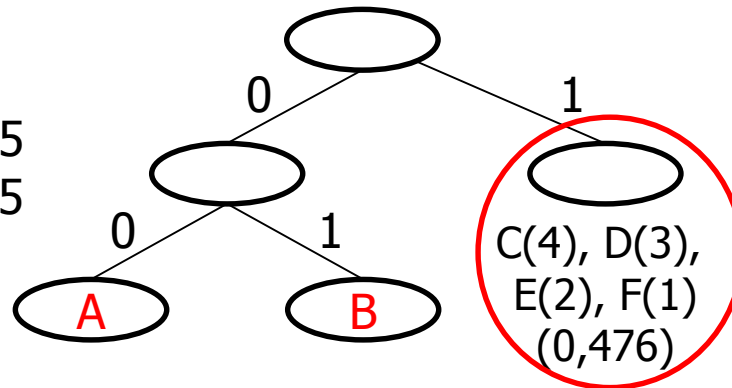
# Codificação de Shannon-Fano

## Exemplo

$$6+5 = 11$$

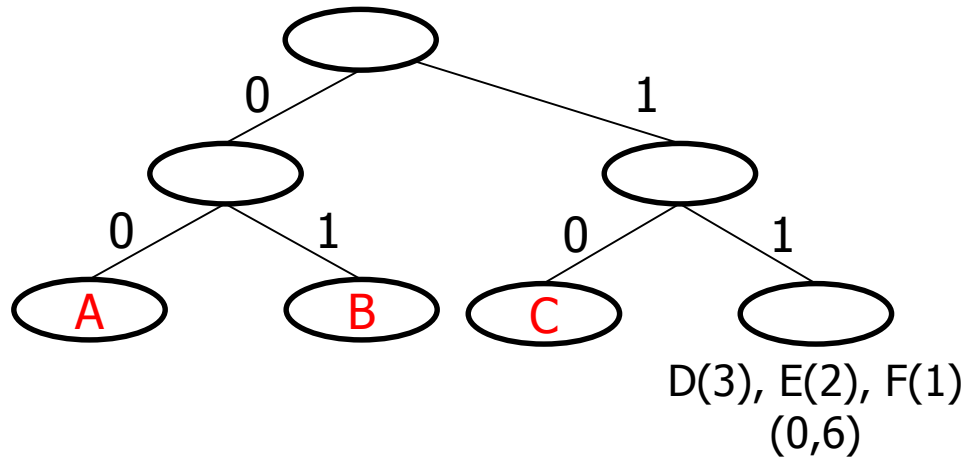
$$6(A) = 0,545$$

$$5(B) = 0,455$$



# Codificação de Shannon-Fano

## Exemplo



$$4+3+2+1 = 10$$

$$4(C) = 0,4$$

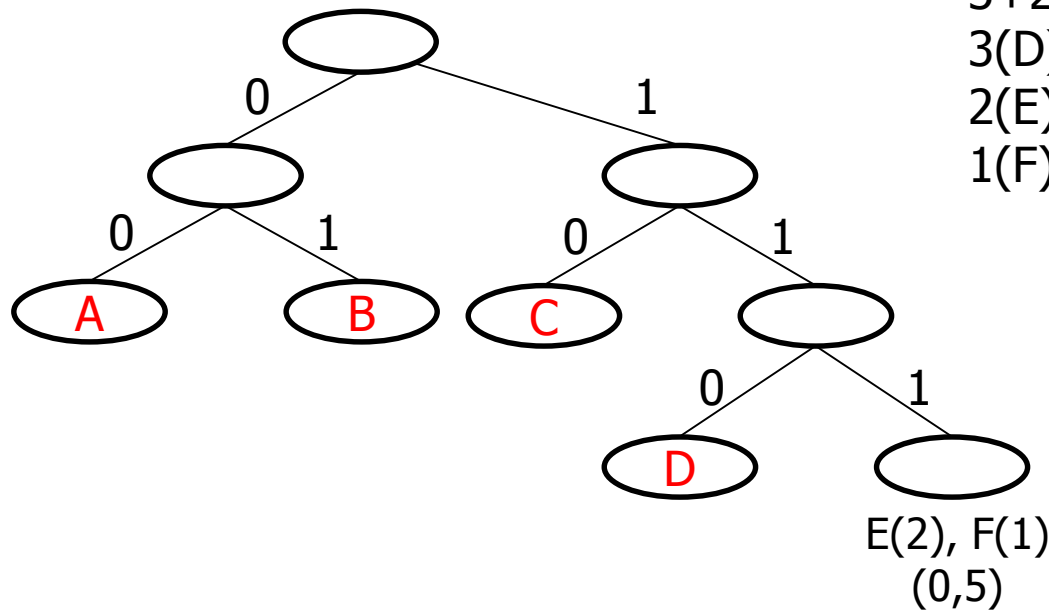
$$3(D) = 0,3$$

$$2(E) = 0,2$$

$$1(F) = 0,1$$

# Codificação de Shannon-Fano

## Exemplo



$$3+2+1 = 6$$

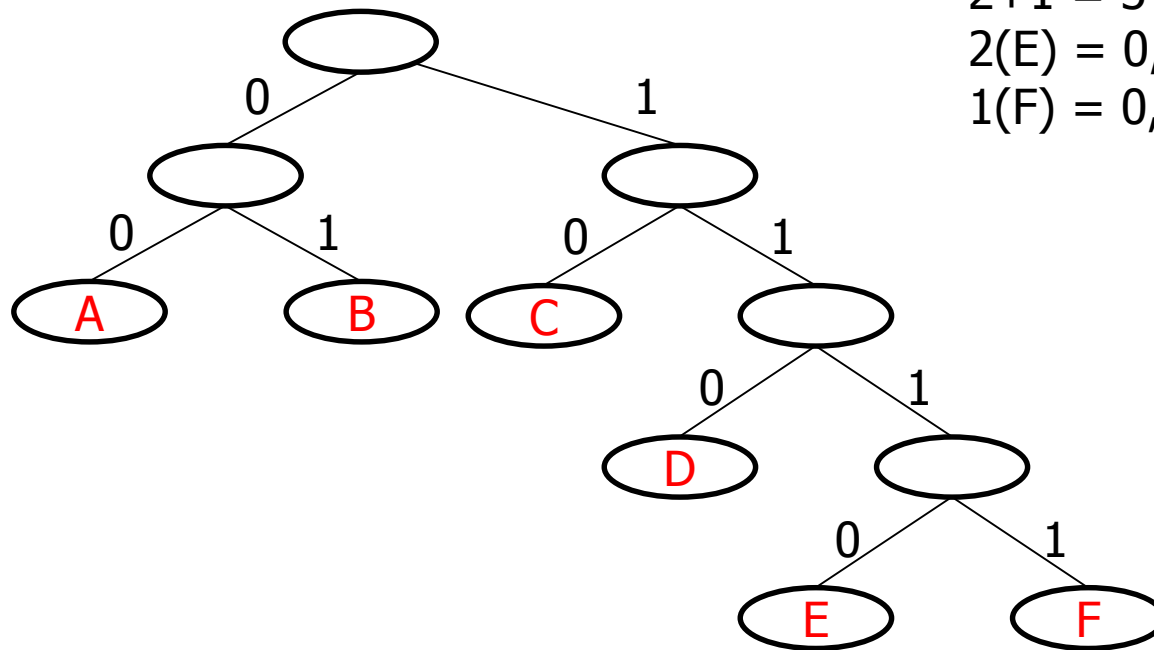
$$3(D) = 0,5$$

$$2(E) = 0,333$$

$$1(F) = 0,167$$

# Codificação de Shannon-Fano

## Exemplo



$$\begin{aligned} 2+1 &= 3 \\ 2(E) &= 0,667 \\ 1(F) &= 0,333 \end{aligned}$$

Codificação Final (51 bits) (contra 168 (21\*8))

00.00.00.00.00.00.01.01.01.01.01.10.10.10.10.110.110.110.1110.1110.1111

# Codificação de Shannon-Fano

## Exemplo

---

- Nesse exemplo a codificação final da sequência tem o mesmo tamanho da obtida pela codificação de Huffman (EXERCÍCIO)
- Em geral o método de Shannon-Fano tem resultados semelhantes ao método de Huffman, podendo ser provado que o tamanho médio dos caracteres,  $T$ , nos dois métodos obedece a seguinte relação
  - $T_H \leq T_{SF} \leq T_H + 1$



# Algoritmos Lempel-Ziv

---





# Algoritmos Lempel-Ziv

---

- Vários métodos de compressão necessitam conhecer as probabilidades dos símbolos que compõem a sequência que se pretende codificar
- Além disso, pode ocorrer de que o processo gerador dos símbolos contenha uma característica dinâmica e as suas estatísticas variarem ao longo do tempo



# Algoritmos Lempel-Ziv

---

- Uma das principais vantagens dos códigos Lempel-Ziv é que os mesmos não requerem informação sobre a distribuição dos dados a codificar, pois baseiam-se no princípio de compressão por substituição
  - LZ77, LZ78, LZW



# Algoritmos Lempel-Ziv

---

- No conjunto de dados a codificar, novas sequências de símbolos são substituídas por referências a ocorrências anteriores da mesma sequência
- Não existe a noção de correspondência entre símbolos (ou conjuntos de símbolos) e palavras de código



# Algoritmos Lempel-Ziv

---

- Nos algoritmos Lempel-Ziv, as sequências são sempre associadas a palavras de código de comprimento constante
- Os dados são sub-divididos num conjunto de sequências – **dicionário** – que vai aumentando à medida que novos dados são analisados
- O dicionário não é enviado explicitamente, sendo construído à medida que são observados novos símbolos tanto no codificador como no decodificador



# LZW (Lempel-Ziv-Welch)

---

- Simétrico, adaptativo e sem perdas
- Formatos que utilizam LZW
  - TIFF, GIF, etc.
- Variante do LZ78



# LZW

---

## Notação

<i>raiz</i>	<i>caractere individual</i>
<i>string</i>	<i>uma sequência de um ou mais caracteres</i>
<i>palavra código</i>	<i>valor associado a uma string</i>
<i>dicionário</i>	<i>tabela que relaciona palavras código e strings</i>
<i>P</i>	<i>string que representa um <b>prefixo</b></i>
<i>C</i>	<i>caractere</i>
<i>cW</i>	<i>palavra código</i>
<i>pW</i>	<i>palavra código que representa um <b>prefixo</b></i>
<i>X &lt;= Y</i>	<i>string X assume o valor da string Y</i>
<i>X+Y</i>	<i>concatenação das string X e Y</i>
<i>string(w)</i>	<i>string correspondente à palavra código w</i>

# LZW

## Codificação

1. No início o dicionário contém todas as raízes possíveis e P é vazio;
2.  $C \leq$  próximo caracter da sequência de entrada;
3. A string  $P+C$  existe no dicionário ?
  - a. se *sim*,
    - i.  $P \leq P+C$ ;
  - b. se *não*,
    - i. coloque a palavra código correspondente a P na sequência codificada;
    - ii. adicione a string  $P+C$  ao dicionário;
    - iii.  $P \leq C$ ;
4. Existem mais caracteres na sequência de entrada ?
  - a. se *sim*,
    - i. volte ao passo 2;
  - b. se *não*,
    - ii. coloque a palavra código correspondente a P na sequência codificada;
    - iii. FIM.

# LZW

## Codificação

1. No início o dicionário contém todas as raízes possíveis e P é vazio;
2.  $C \leq$  próximo caracter da sequência de entrada;
3. A string P+C existe no dicionário ?
  - a. se *sim*,
    - i.  $P \leq P+C$ ;
  - b. se *não*,
    - i. coloque a palavra código correspondente a P na sequência codificada;
    - ii. adicione a string P+C ao dicionário;
    - iii.  $P \leq C$ ;
4. Existem mais caracteres na sequência de entrada ?
  - a. se *sim*,
    - i. volte ao passo 2;
  - b. se *não*,
    - ii. coloque a palavra código correspondente a P na sequência codificada;
    - iii. FIM.

Posição	1	2	3	4	5	6	7	8	9
Caracter	a	b	b	a	b	a	b	a	c

1.  $SC = \emptyset$  //SC = sequência codificada
2. Inicialização do dicionário com as raízes:

Palavra Código	String
1	a
2	b
3	c

3.  $P = \emptyset$
4.  $C = 'a'$  (posição 1)
5.  $P+C = 'a'$  existe no dicionário
6.  $P = P+C = 'a'$



# LZW

## Codificação

1. No início o dicionário contém todas as raízes possíveis e P é vazio;
2.  $C \leq$  próximo caracter da sequência de entrada;
3. A string P+C existe no dicionário ?
  - a. se *sim*,
    - i.  $P \leq P+C$ ;
  - b. se *não*,
    - i. coloque a palavra código correspondente a P na sequência codificada;
    - ii. adicione a string P+C ao dicionário;
    - iii.  $P \leq C$ ;
4. Existem mais caracteres na sequência de entrada ?
  - a. se *sim*,
    - i. volte ao passo 2;
  - b. se *não*,
    - ii. coloque a palavra código correspondente a P na sequência codificada;
    - iii. FIM.

7.  $C = 'b'$  (próximo caracter da sequência de entrada – posição 2)
8.  $P+C = 'ab'$  não existe no dicionário
9.  $SC = 1$  (corresponde a  $P = 'a'$ )
- 10.

Palavra Código	String
1	a
2	b
3	c
4	ab

11.  $P = C = 'b'$

# LZW

## Codificação

1. No início o dicionário contém todas as raízes possíveis e P é vazio;
2.  $C \leq$  próximo caracter da sequência de entrada;
3. A string  $P+C$  existe no dicionário ?
  - a. se *sim*,
    - i.  $P \leq P+C$ ;
  - b. se *não*,
    - i. coloque a palavra código correspondente a P na sequência codificada;
    - ii. adicione a string  $P+C$  ao dicionário;
    - iii.  $P \leq C$ ;
4. Existem mais caracteres na sequência de entrada ?
  - a. se *sim*,
    - i. volte ao passo 2;
  - b. se *não*,
    - ii. coloque a palavra código correspondente a P na sequência codificada;
    - iii. FIM.

12.  $C = 'b'$  (próximo caracter da sequência de entrada – posição 3)

13.  $P+C = 'bb'$  não existe no dicionário

14.  $SC = 1\ 2$  (corresponde a  $P = 'b'$ )

15.

Palavra Código	String
1	a
2	b
3	c
4	ab
5	bb

16.  $P = C = 'b'$

# LZW

## Codificação

1. No início o dicionário contém todas as raízes possíveis e P é vazio;
2.  $C \leq$  próximo caracter da sequência de entrada;
3. A string P+C existe no dicionário ?
  - a. se *sim*,
    - i.  $P \leq P+C$ ;
  - b. se *não*,
    - i. coloque a palavra código correspondente a P na sequência codificada;
    - ii. adicione a string P+C ao dicionário;
    - iii.  $P \leq C$ ;
4. Existem mais caracteres na sequência de entrada ?
  - a. se *sim*,
    - i. volte ao passo 2;
  - b. se *não*,
    - ii. coloque a palavra código correspondente a P na sequência codificada;
    - iii. FIM.

17.  $C = 'a'$  (próximo caracter da sequência de entrada – posição 4)

18.  $P+C = 'ba'$  não existe no dicionário

19.  $SC = 1\ 2\ 2$  (corresponde a  $P = 'b'$ )

20.

Palavra Código	String
1	a
2	b
3	c
4	ab
5	bb
6	ba

21.  $P = C = 'a'$

# LZW

## Codificação

1. No início o dicionário contém todas as raízes possíveis e P é vazio;
2.  $C \leq$  próximo caracter da sequência de entrada;
3. A string  $P+C$  existe no dicionário ?
  - a. se *sim*,
    - i.  $P \leq P+C$ ;
  - b. se *não*,
    - i. coloque a palavra código correspondente a P na sequência codificada;
    - ii. adicione a string  $P+C$  ao dicionário;
    - iii.  $P \leq C$ ;
4. Existem mais caracteres na sequência de entrada ?
  - a. se *sim*,
    - i. volte ao passo 2;
  - b. se *não*,
    - ii. coloque a palavra código correspondente a P na sequência codificada;
    - iii. FIM.

22.  $C = 'b'$  (próximo caracter da sequência de entrada – posição 5)

23.  $P+C = 'ab'$  existe no dicionário

24.  $P = P+C = 'ab'$

25.  $C = 'a'$  (próximo caracter da sequência de entrada – posição 6)

26.  $P+C = 'aba'$  não existe no dicionário

27.  $SC = 1\ 2\ 2\ 4$  (corresponde a  $P = 'ab'$ )

Palavra Código	String
1	a
2	b
3	c
4	ab
5	bb
6	ba
7	aba

28.  $P = C = 'a'$

# LZW

## Codificação

1. No início o dicionário contém todas as raízes possíveis e P é vazio;
2.  $C \leq$  próximo caracter da sequência de entrada;
3. A string  $P+C$  existe no dicionário ?
  - a. se *sim*,
    - i.  $P \leq P+C$ ;
  - b. se *não*,
    - i. coloque a palavra código correspondente a P na sequência codificada;
    - ii. adicione a string  $P+C$  ao dicionário;
    - iii.  $P \leq C$ ;
4. Existem mais caracteres na sequência de entrada ?
  - a. se *sim*,
    - i. volte ao passo 2;
  - b. se *não*,
    - ii. coloque a palavra código correspondente a P na sequência codificada;
    - iii. FIM.

29.  $C = 'b'$  (próximo caracter da sequência de entrada – posição 7)

30.  $P+C = 'ab'$  existe no dicionário

31.  $P = P+C = 'ab'$

---

32.  $C = 'a'$  (próximo caracter da sequência de entrada – posição 8)

33.  $P+C = 'aba'$  existe no dicionário

# LZW

## Codificação

1. No início o dicionário contém todas as raízes possíveis e P é vazio;
2.  $C \leq$  próximo caracter da sequência de entrada;
3. A string  $P+C$  existe no dicionário ?
  - a. se *sim*,
    - i.  $P \leq P+C$ ;
  - b. se *não*,
    - i. coloque a palavra código correspondente a P na sequência codificada;
    - ii. adicione a string  $P+C$  ao dicionário;
    - iii.  $P \leq C$ ;
4. Existem mais caracteres na sequência de entrada ?
  - a. se *sim*,
    - i. volte ao passo 2;
  - b. se *não*,
    - ii. coloque a palavra código correspondente a P na sequência codificada;
    - iii. FIM.

34.  $C = 'c'$  (próximo caracter da sequência de entrada – posição 9)

35.  $P+C = 'abac'$  não existe no dicionário

36.  $S = 1\ 2\ 2\ 4\ 7$  (corresponde a  $P = 'aba'$ )

37.

Palavra Código	String
1	a
2	b
3	c
4	ab
5	bb
6	ba
7	aba
8	abac

a.b.b.a.b.a.b.a.c  
9 bytes (**72 bits**)

1.2.2.4.7.3  
6 bytes (**48 bits**)

8 = 1000 (4 bits)  
 $6 * 4 =$  **24 bits**  
3 bytes

38.  $P = C = 'c'$

39. Não existem mais caracteres a codificar, logo,  $SC = 1\ 2\ 2\ 4\ 7\ 3$   
(correspondente a  $P = 'c'$ )

40. FIM

# LZW

## Decodificação

1. No início o dicionário contém todas as raízes possíveis;
2.  $cW \leq$  primeira palavra código na sequência codificada (sempre é uma raiz);
3. Coloque a  $string(cW)$  na sequência de saída;
4.  $pW \leq cW$ ;
5.  $cW \leq$  próxima palavra código da sequência codificada;
6. A  $string(cW)$  existe no dicionário ?
  - a. se *sim*,
    - i. coloque a  $string(cW)$  na sequência de saída;
    - ii.  $P \leq string(pW)$ ;
    - iii.  $C \leq$  primeiro caracter da  $string(cW)$ ;
    - iv. adicione a  $string P+C$  ao dicionário;
  - b. se *não*,
    - i.  $P \leq string(pW)$ ;
    - ii.  $C \leq$  primeiro caracter da  $string(pW)$ ;
    - iii. coloque a  $string P+C$  na sequência de saída e adicione-a ao dicionário;
7. Existem mais palavras código na sequência codificada ?
  - a. se *sim*,
    - i. volte ao passo 4;
  - b. se *não*,
    - i. FIM.

# LZW

## Decodificação

Posição	1	2	3	4	5	6
Palavra Código	1	2	2	4	7	3

1. No início o dicionário contém todas as raízes possíveis;
2.  $cW \leq$  primeira palavra código na sequência codificada (sempre é uma raiz);
3. Coloque a  $string(cW)$  na sequência de saída;
4.  $pW \leq cW$ ;
5.  $cW \leq$  próxima palavra código da sequência codificada;
6. A  $string(cW)$  existe no dicionário ?
  - a. se *sim*,
    - i. coloque a  $string(cW)$  na sequência de saída;
    - ii.  $P \leq string(pW)$ ;
    - iii.  $C \leq$  primeiro caracter da  $string(cW)$ ;
    - iv. adicione a  $string P+C$  ao dicionário;
  - b. se *não*,
    - i.  $P \leq string(pW)$ ;
    - ii.  $C \leq$  primeiro caracter da  $string(pW)$ ;
    - iii. coloque a  $string P+C$  na sequência de saída e adicione-a ao dicionário;
7. Existem mais palavras código na sequência codificada ?
  - a. se *sim*,
    - i. volte ao passo 4;
  - b. se *não*,
    - i. FIM.

1.  $SS = \emptyset$  //SS = sequência saída
2. Inicialização do dicionário com as raízes:

Palavra Código	String
1	a
2	b
3	c

3.  $cW = 1$  (primeira palavra código)
4.  $SS = a$  (corresponde a  $string(cW)$  )



# LZW

## Decodificação

5.  $pW = cW = 1$
6.  $cW = 2$  (próxima palavra código da sequência codificada – posição 2)
7.  $string(cW) = 'b'$  existe no dicionário
8.  $SS = a\ b$  (corresponde a  $string(cW)$  )
9.  $P = string(pW) = 'a'$
10.  $C = 1^o$  caracter da  $string(cW) = 'b'$
11.  $P+C = 'ab'$

Palavra Código	String
1	a
2	b
3	c
4	ab

1. No início o dicionário contém todas as raízes possíveis;
2.  $cW \leq$  primeira palavra código na sequência codificada (sempre é uma raiz);
3. Coloque a  $string(cW)$  na sequência de saída;
4.  $pW \leq cW$ ;
5.  $cW \leq$  próxima palavra código da sequência codificada;
6. A  $string(cW)$  existe no dicionário ?
  - a. se *sim*,
    - i. coloque a  $string(cW)$  na sequência de saída;
    - ii.  $P \leq string(pW)$ ;
    - iii.  $C \leq$  primeiro caracter da  $string(cW)$ ;
    - iv. adicione a  $string P+C$  ao dicionário;
  - b. se *não*,
    - i.  $P \leq string(pW)$ ;
    - ii.  $C \leq$  primeiro caracter da  $string(pW)$ ;
    - iii. coloque a  $string P+C$  na sequência de saída e adicione-a ao dicionário;
7. Existem mais palavras código na sequência codificada ?
  - a. se *sim*,
    - i. volte ao passo 4;
  - b. se *não*,
    - i. FIM.

# LZW

## Decodificação

12.  $pW = cW = 2$

13.  $cW = 2$  (próxima palavra código da sequência codificada – posição 3)

14.  $\text{string}(cW) = 'b'$  existe no dicionário

15.  $SS = a\ b\ b$  (corresponde a  $\text{string}(cW)$  )

16.  $P = \text{string}(pW) = 'b'$

17.  $C = 1^{\circ}$  caracter da  $\text{string}(cW) = 'b'$

18.  $P+C = 'bb'$

1. No início o dicionário contém todas as raízes possíveis;
2.  $cW \leq$  primeira palavra código na sequência codificada (sempre é uma raiz);
3. Coloque a  $\text{string}(cW)$  na sequência de saída;
4.  $pW \leq cW$ ;
5.  $cW \leq$  próxima palavra código da sequência codificada;
6. A  $\text{string}(cW)$  existe no dicionário ?
  - a. se *sim*,
    - i. coloque a  $\text{string}(cW)$  na sequência de saída;
    - ii.  $P \leq \text{string}(pW)$ ;
    - iii.  $C \leq$  primeiro caracter da  $\text{string}(cW)$ ;
    - iv. adicione a  $\text{string } P+C$  ao dicionário;
  - b. se *não*,
    - i.  $P \leq \text{string}(pW)$ ;
    - ii.  $C \leq$  primeiro caracter da  $\text{string}(pW)$ ;
    - iii. coloque a  $\text{string } P+C$  na sequência de saída e adicione-a ao dicionário;
7. Existem mais palavras código na sequência codificada ?
  - a. se *sim*,
    - i. volte ao passo 4;
  - b. se *não*,
    - i. FIM.

Palavra Código	String
1	a
2	b
3	c
4	ab
5	bb

# LZW

## Decodificação

19.  $pW = cW = 2$

20.  $cW = 4$  (próxima palavra código da sequência codificada – posição 4)

21.  $\text{string}(cW) = \text{'ab'}$  existe no dicionário

22.  $SS = a\ b\ b\ a\ b$  (corresponde a  $\text{string}(cW)$  )

23.  $P = \text{string}(pW) = \text{'b'}$

24.  $C = 1^{\circ}$  caracter da string ( $cW$ ) = 'a'

25.  $P+C = \text{'ba'}$

1. No início o dicionário contém todas as raízes possíveis;
2.  $cW \leq$  primeira palavra código na sequência codificada (sempre é uma raiz);
3. Coloque a  $\text{string}(cW)$  na sequência de saída;
4.  $pW \leq cW$ ;
5.  $cW \leq$  próxima palavra código da sequência codificada;
6. A  $\text{string}(cW)$  existe no dicionário ?
  - a. se *sim*,
    - i. coloque a  $\text{string}(cW)$  na sequência de saída;
    - ii.  $P \leq \text{string}(pW)$ ;
    - iii.  $C \leq$  primeiro caracter da  $\text{string}(cW)$ ;
    - iv. adicione a string  $P+C$  ao dicionário;
  - b. se *não*,
    - i.  $P \leq \text{string}(pW)$ ;
    - ii.  $C \leq$  primeiro caracter da  $\text{string}(pW)$ ;
    - iii. coloque a string  $P+C$  na sequência de saída e adicione-a ao dicionário;
7. Existem mais palavras código na sequência codificada ?
  - a. se *sim*,
    - i. volte ao passo 4;
  - b. se *não*,
    - i. FIM.

Palavra Código	String
1	a
2	b
3	c
4	ab
5	bb
6	ba

# LZW

## Decodificação

26.  $pW = cW = 4$

27.  $cW = 7$  (próxima palavra código da sequência codificada – posição 5)

28.  $string(cW) = ?$  não existe no dicionário

29.  $P = string(pW) = 'ab'$

30.  $C = 1^o$  caracter da string  $(pW) = 'a'$

31.  $SS = a\ b\ b\ a\ b\ a\ b\ a$  (correspondente a  $P+C = 'aba'$ )

32.  $P+C = 'aba'$

1. No início o dicionário contém todas as raízes possíveis;
2.  $cW \leq$  primeira palavra código na sequência codificada (sempre é uma raiz);
3. Coloque a  $string(cW)$  na sequência de saída;
4.  $pW \leq cW$ ;
5.  $cW \leq$  próxima palavra código da sequência codificada;
6. A  $string(cW)$  existe no dicionário ?
  - a. se *sim*,
    - i. coloque a  $string(cW)$  na sequência de saída;
    - ii.  $P \leq string(pW)$ ;
    - iii.  $C \leq$  primeiro caracter da  $string(cW)$ ;
    - iv. adicione a string  $P+C$  ao dicionário;
  - b. se *não*,
    - i.  $P \leq string(pW)$ ;
    - ii.  $C \leq$  primeiro caracter da  $string(pW)$ ;
    - iii. coloque a string  $P+C$  na sequência de saída e adicione-a ao dicionário;
7. Existem mais palavras código na sequência codificada ?
  - a. se *sim*,
    - i. volte ao passo 4;
  - b. se *não*,
    - i. FIM.

Palavra Código	String
1	a
2	b
3	c
4	ab
5	bb
6	ba
7	aba

# LZW

## Decodificação

1. No início o dicionário contém todas as raízes possíveis;
2.  $cW \leq$  primeira palavra código na sequência codificada (sempre é uma raiz);
3. Coloque a  $string(cW)$  na sequência de saída;
4.  $pW \leq cW$ ;
5.  $cW \leq$  próxima palavra código da sequência codificada;
6. A  $string(cW)$  existe no dicionário ?
  - a. se *sim*,
    - i. coloque a  $string(cW)$  na sequência de saída;
    - ii.  $P \leq string(pW)$ ;
    - iii.  $C \leq$  primeiro caracter da  $string(cW)$ ;
    - iv. adicione a  $string P+C$  ao dicionário;
  - b. se *não*,
    - i.  $P \leq string(pW)$ ;
    - ii.  $C \leq$  primeiro caracter da  $string(pW)$ ;
    - iii. coloque a  $string P+C$  na sequência de saída e adicione-a ao dicionário;
7. Existem mais palavras código na sequência codificada ?
  - a. se *sim*,
    - i. volte ao passo 4;
  - b. se *não*,
    - i. FIM.

33.  $pW = cW = 7$

34.  $cW = 3$  (próxima palavra código da sequência codificada – posição 6)

35.  $string(cW) = 'c'$  existe no dicionário

36.  $SS = a b b a b a b a c$  (correspondente a  $string(cW)$  )

37.  $P = string(pW) = 'aba'$

38.  $C = 1^o$  caracter da  $string(cW) = 'c'$

39.  $P+C = 'abac'$

Palavra Código	String
1	a
2	b
3	c
4	ab
5	bb
6	ba
7	aba
8	abac

40. FIM

# Exercício

## LZW

---

- Considere como raízes do dicionário a tabela ASCII (0..255)
- Codifique a sequência "A\_ASA\_DA\_CASA"
  - Qual a SC?
  - Quais são as novas palavras do dicionário?
    - Mostrar todas a partir do código 256

Quantos bits são necessários para codificar a sequência?

Observação: Códigos ASCII  
A=65 / \_=95 / S=83 / D=68 / C=67



# LZ77

---

- LZ77 não tem patente, razão pela qual é usado em muitos compactadores (LZ78 e LZW possuem patente)
- Devido a grande diferença entre a compressão e a descompressão o algoritmo é assimétrico



# LZ77

---

- A variante mais comum do LZ77 é conhecida como DEFLATE e combina o uso de LZ77 com o Código de Huffman





# LZ77

---

- Entre os programas e formatos que usam LZ e variantes temos:
  - O LZ77 é usado no PKZIP, GZIP e no formato de imagens PNG
  - Winzip, Winrar usa o LZ77 e o Huffman

# LZ77

## Codificação

---

- São necessárias duas estruturas
  - Janela (Dicionário)
    - Representa as partes do arquivo que já foram lidas
    - Tem tamanho definido, permitindo que os dados sejam enfileirados dentro dela, eliminando os bytes mais antigos quando seu limite de tamanho é atingido

# LZ77

## Codificação

---

- São necessárias duas estruturas (cont.)
  - Buffer
    - Representa o que ainda será lido e processado pelo algoritmo
    - Também tem tamanho definido, em geral, dezenas de vezes menor que a janela

# LZ77

## Codificação

---

Inicialize o Buffer

[1] Encontre a maior sequência existente na janela que case exatamente com o início da sequência existente no buffer

# LZ77

## Codificação

- [2] Ao encontrar tal sequência emita como saída a tupla  $(S_{pos}, S_{tam}, c)$
- $S_{pos}$ : posição da sequência casada dentro da janela (contada, em geral, de trás para frente)
  - $S_{tam}$ : tamanho dessa sequência
  - $c$ : próximo caractere presente no buffer depois dessa sequência
  - No caso de não ser encontrado nenhum casamento dentro da janela, emita a tupla  $(0,0,c)$ , indicando que houve um "casamento" de tamanho 0, e transfira apenas o caractere  $c$  para o buffer

**(Cont.)**

# LZ77

## Codificação

[3] Transfira toda a sequência casada, mais o caractere extra, para a janela, que terá seus elementos mais antigos removidos (caso esteja cheia) e preencha o buffer com novos dados lidos do arquivo

[4] Continue o processo até o final do arquivo

**(Cont.)**

# LZ77

## Decodificação

---

Copie os caracteres da tupla, nas posições e quantidades indicadas, para a saída, e acrescente o caractere c, repetindo o processo até o fim das tuplas

# LZ77

## Exemplo

---

- Comprimir: A\_ASA\_DA\_CASA
  - Janela: 8
  - Buffer: 4

INICIO DA CODIFICAÇÃO



# LZ77

## Exemplo

---

A\_ASA\_DA\_CASA

Janela (já processado)

--	--	--	--	--	--	--	--

Buffer (a ser processado)

--	--	--	--

# LZ77

## Exemplo

A\_ASA\_DA\_CASA

Janela (já processado)

--	--	--	--	--	--	--	--

Buffer (a ser processado)

A	_	A	S
---	---	---	---

Existe subsequência? (não)

# LZ77

## Exemplo

---

A\_ASA\_DA\_CASA

Janela (já processado)

A							
---	--	--	--	--	--	--	--

Buffer (a ser processado)

_	A	S	A
---	---	---	---

Saída: (0,0,A);

# LZ77

## Exemplo

A\_ASA\_DA\_CASA

Janela (já processado)

A							
---	--	--	--	--	--	--	--

Buffer (a ser processado)

_	A	S	A
---	---	---	---

Existe subsequência? (não)

Saída: (0,0,A);

# LZ77

## Exemplo

A\_ASA\_DA\_CASA

Janela (já processado)

A	_						
---	---	--	--	--	--	--	--

Buffer (a ser processado)

A	S	A	_
---	---	---	---

Saída: (0,0,A); (0,0,\_);

# LZ77

## Exemplo

A\_ASA\_DA\_CASA

Janela (já processado)

A	_						
---	---	--	--	--	--	--	--

Buffer (a ser processado)

A	S	A	_
---	---	---	---

Existe subsequência? (sim)

Saída: (0,0,A); (0,0,\_);

# LZ77

## Exemplo

A\_ASA\_DA\_CASA

Janela (já processado)

A	_	A	S				
---	---	---	---	--	--	--	--

Buffer (a ser processado)

A	_	D	A
---	---	---	---

Saída: (0,0,A); (0,0,\_); (2,1,S);

# LZ77

## Exemplo

A\_ASA\_DA\_CASA

Janela (já processado)

A	_	A	S				
---	---	---	---	--	--	--	--

Buffer (a ser processado)

A	_	D	A
---	---	---	---

Existe subsequência? (sim)

Saída: (0,0,A); (0,0,\_); (2,1,S);



# LZ77

## Exemplo

A\_ASA\_DA\_CASA

Janela (já processado)

A	_	A	S	A	_	D	
---	---	---	---	---	---	---	--

Buffer (a ser processado)

A	_	C	A
---	---	---	---

Saída: (0,0,A); (0,0,\_); (2,1,S); (4,2,D);

# LZ77

## Exemplo

A\_ASA\_DA\_CASA

Janela (já processado)

A	_	A	S	A	_	D	
---	---	---	---	---	---	---	--

Buffer (a ser processado)

A	_	C	A
---	---	---	---

Existe subsequência? (sim)

Saída: (0,0,A); (0,0,\_); (2,1,S); (4,2,D);

# LZ77

## Exemplo

A\_ASA\_DA\_CASA

Janela (já processado)

A	S	A	_	D	A	_	C	Sai (A, _)
---	---	---	---	---	---	---	---	------------

Buffer (a ser processado)

A	S	A	
---	---	---	--

Saída: (0,0,A); (0,0,\_); (2,1,S); (4,2,D); (3,2,C);

# LZ77

## Exemplo

A\_ASA\_DA\_CASA

Janela (já processado)

A	S	A	_	D	A	_	C	Sai (A, _)
---	---	---	---	---	---	---	---	------------

Buffer (a ser processado)

A	S	A	
---	---	---	--

Existe subsequência? (sim)

Saída: (0,0,A); (0,0,\_); (2,1,S); (4,2,D); (3,2,C);

# LZ77

## Exemplo

Cada tupla ocupa 15 bits (4 para a posição dentro da janela (8=1000), 3 para o tamanho (4=100) e 8 para o caractere no final). Total = 90 bits (6\*15)

A\_ASA\_DA\_CASA

Janela (já processado)

_	D	A	_	C	A	S	A
---	---	---	---	---	---	---	---

Sai (A, s, A)

Buffer (a ser processado)

--	--	--	--

Saída: (0,0,A); (0,0,\_); (2,1,S); (4,2,D); (3,2,C); (8,3,EOF)

FIM DA CODIFICAÇÃO

# LZ77

## Exemplo

---

Saída: (0,0,A); (0,0,\_); (2,1,S); (4,2,D); (3,2,C); (8,3,EOF)

A

INÍCIO DA DECODIFICAÇÃO

# LZ77

## Exemplo

---

Saída: (0,0,A); (0,0,\_); (2,1,S); (4,2,D); (3,2,C); (8,3,EOF)

A\_

# LZ77

## Exemplo

---

Saída: (0,0,A); (0,0,\_); (2,1,S); (4,2,D); (3,2,C); (8,3,EOF)

A\_AS



# LZ77

## Exemplo

---

Saída: (0,0,A); (0,0,\_); (2,1,S); (4,2,D); (3,2,C); (8,3,EOF)

A\_ASA\_D

# LZ77

## Exemplo

---

Saída: (0,0,A); (0,0,\_); (2,1,S); (4,2,D); (3,2,C); (8,3,EOF)

A\_ASA\_DA\_C

# LZ77

## Exemplo

---

Saída: (0,0,A); (0,0,\_); (2,1,S); (4,2,D); (3,2,C); (8,3,EOF)

A\_ASA\_DA\_CASA

FIM DA DECODIFICAÇÃO



# LZ77

---

- O tamanho da janela e do buffer impactam diretamente na performance do compressor
  - Quanto maior eles forem, melhor a compressão, mas também mais lenta ela fica
  - O tamanho dessas estruturas deve ser bem estudado quando da implementação desse algoritmo



# LZ77

---

- A dimensão da janela (dicionário) condiciona até onde se pode pesquisar
- A dimensão do buffer condiciona a máxima dimensão da sequência a codificar

# Exercício

## LZ77

---

- Comprimir
  - abbababac
  - Comparar com LZW

Quantos bits são necessários para codificar a sequência?



# Referências

---

- Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; Stein, C. Introduction to algorithms. 2001.
- Tenenbaum, A. M.; Langsan, Y.; Augenstein, M. Estruturas de dados usando C. 1995.
- Pu, I. M. Fundamental data compression. 2006.
- Sayood, K. Introduction to data compression. 1998.
- Salomon, D. Data compression: the complete reference. 2000.



# Referências Web

---

- [www.decom.fee.unicamp.br/dspcom/EE088/Algoritmo\\_LZW.pdf](http://www.decom.fee.unicamp.br/dspcom/EE088/Algoritmo_LZW.pdf)
- <http://pt.wikipedia.org/wiki/LZW>
- <http://pt.wikipedia.org/wiki/LZ77>
- [http://pt.wikipedia.org/wiki/Codifica%C3%A7%C3%A3o de Shannon-Fano](http://pt.wikipedia.org/wiki/Codifica%C3%A7%C3%A3o_de_Shannon-Fano)
- <http://www.binaryessence.com/> (não explorada)
- <http://www.algoanim.net/> (não explorada)