Codificação e Compressão

Aula 5 – Métodos de Dicionário

Prof. Adão Antonio de Souza Junior - IFSul

Nas aula passada

1. Alguns elementos de codificação MATLAB
    1. Organização em blocos
    2. Varredura de blocos para maximizar RLE
2. Codificação aritmética
    1. Ideia fundamental por trás da codificação aritmética
    2. Algoritmos de codificação e decodificação
    3. Conversão do número para a mensagem em bits

Fluxos de codificação (cont.)

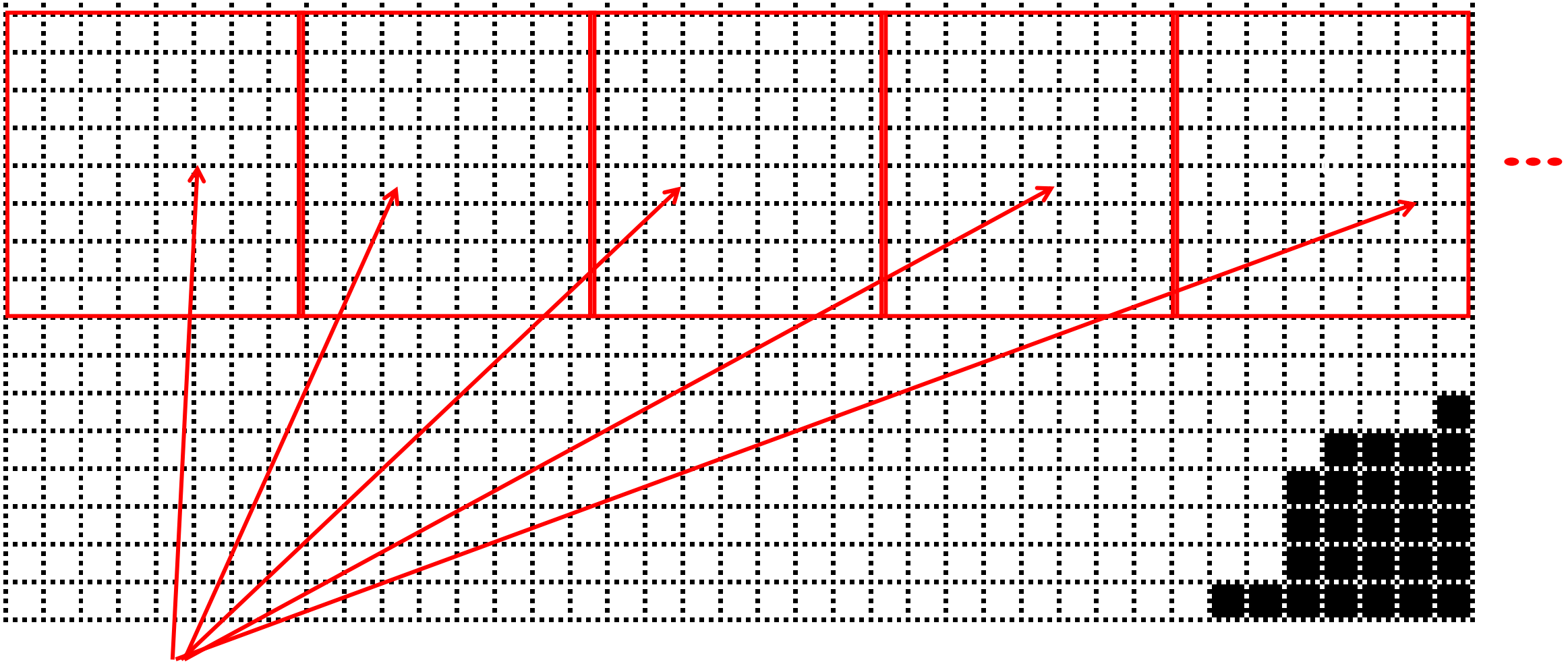
- No caso de imagens o fluxo é mais complexo:
  - Imagens são bidimensionais, com um terceiro índice sendo usado para representar as três componentes de cor em imagens coloridas (caso default)
  - Cada pixel de uma imagem é representado assim por três informações (R,G e B; Y, Cb e Cr, ...) cada uma dessas representada por um número inteiro com número de bits dado pela resolução da imagem.
  - Nos exemplos dados as imagens lidas são formadas por três matrizes correspondendo as componentes R, G e B de cada ponto.
  - As dimensões h (altura) e w (largura) da imagem são as mesmas dimensões das matrizes lidas.

Ordem de leitura da imagem

- Em geral, no entanto, não se lê a imagem inteira pixel a pixel fazendo a varredura de cima para baixo
  - Primeiro se divide a imagem em blocos (as vezes chamadas macrocelulas) que podem ser 2x2, 4x4, 8x8, 16x16 e outras combinações
- Esses blocos são lidos linha (de blocos) a linha (de blocos) da esquerda para a direita e de cima para baixo
- Dentro de cada bloco, no entanto, há uma sequencia de leitura também, chamada scan

Ordem de leitura da imagem

- Em geral, no entanto, não se lê a imagem inteira pixel a pixel fazendo a varredura de cima para baixo
  - Primeiro se divide a imagem em blocos (as vezes chamadas macrocelulas) que podem ser 2x2, 4x4, 8x8, 16x16 e outras combinações
- Esses blocos são lidos linha (de blocos) a linha (de blocos) da esquerda para a direita e de cima para baixo
- Dentro de cada bloco, no entanto, há uma sequencia de leitura também, chamada scan

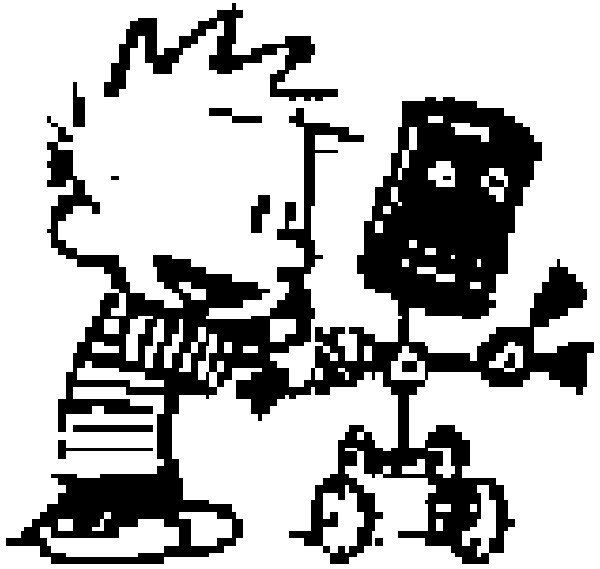


Primeira linha de blocos

Organização de cor

- A imagem é armazenada em conjuntos de matrizes com as dimensões dadas pela largura x altura x numero de componentes
- Quando a imagem é colorida ela é composta por três imagens
  - RGB : Uma para cada componente de cor (não separa luminância)
  - YCbCr : Mais usado em compressão pois a primeira tem a informação de luminância (intensidade da cor) e as duas demais da cor em si.
- O MATLAB permite facilmente tornar matrizes em vetores a questão é da organização que se deseja e de se manter esse processo reversível

98 x 98 (1bit)

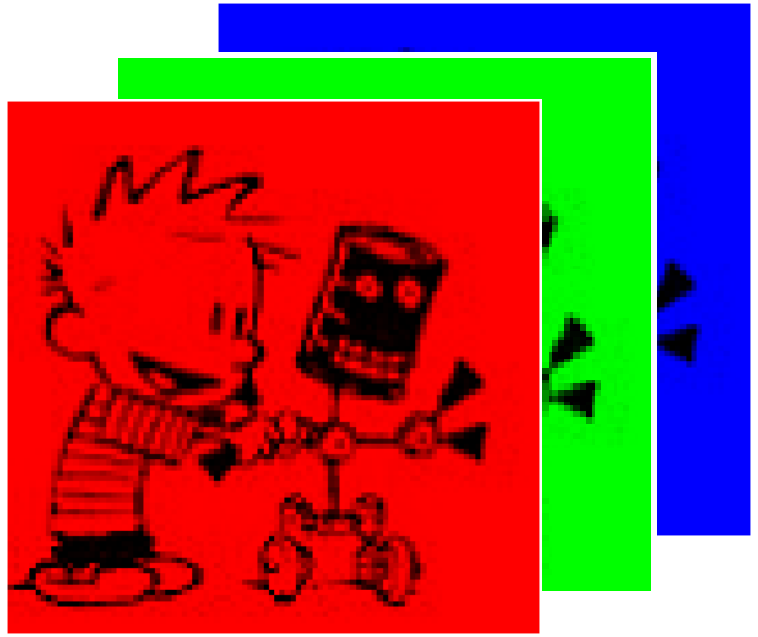


MONOCROMATICA

98 x 98 (8 bits)

GRAYSCALE

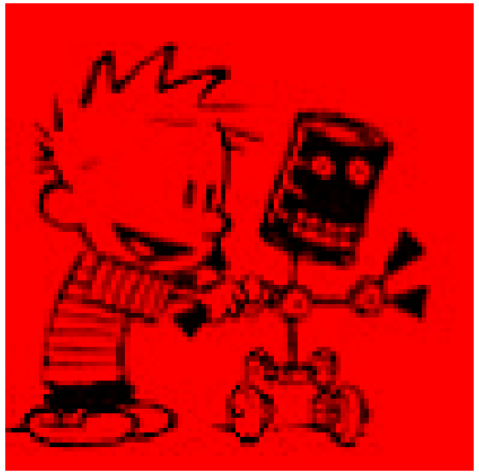
98 x 98 x 3 (8 bits)

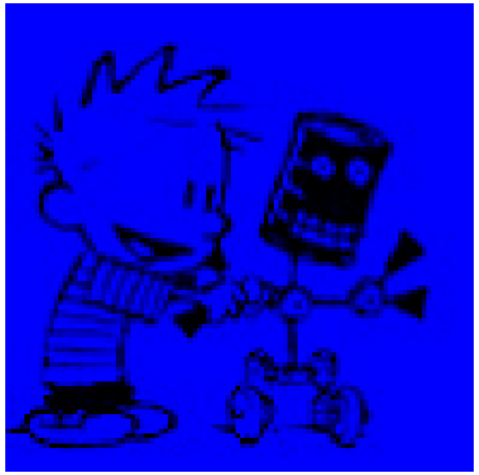


COLORIDA (RGB)

Conversão RGB → YCbCr

Focamos no RGB / Toma o mesmo vale p/ qq blocs





RGB

YCbCr

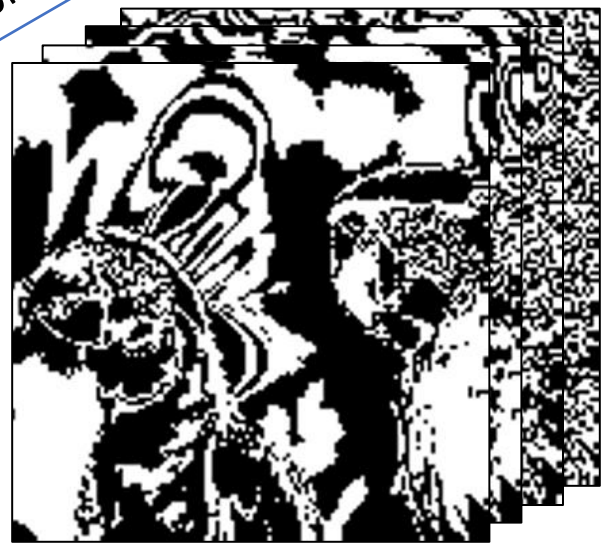
Uso de bitplanes ou quantização

- Pensando nos bits que compõe a informação de cada ponto, os mais significativos apresentam mais informação e os menos significativos muitas vezes são aproximadamente aleatórios
- Quando se organiza a imagem em bitplanes fica fácil ver isso
- Ao se reorganizar a imagem se pode fazer isso de duas formas:
  - Sem perdas (usando todos os bitplanes)
  - Com perdas (descartando os bitplanes menos significativos (zeros)
- Note que, uma segunda forma de fazer codificação com perdas é a quantização, que será vista na segunda área.

Exemplo de bitplanes

Imagem em tons de cinza (8 bits p/ pixel)

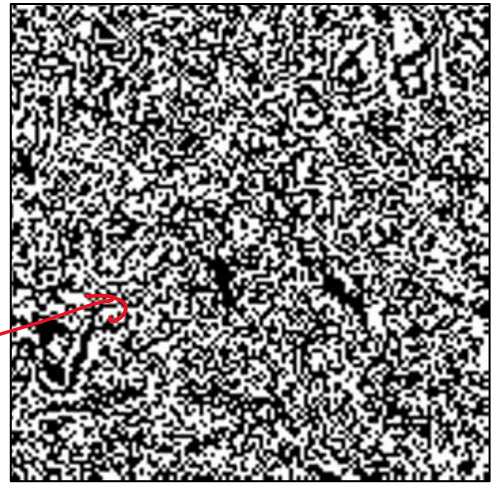
Oito planos de bit



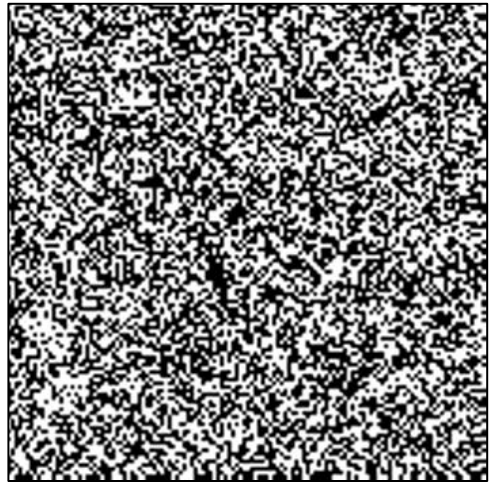
bits - são quase
↳ tem
informação (pode variar?)

Plano 8

Plano 7



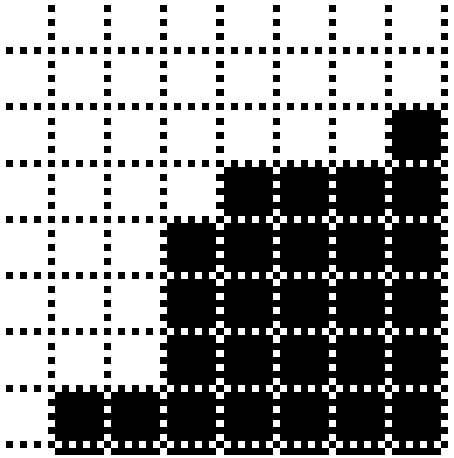
Plano 2

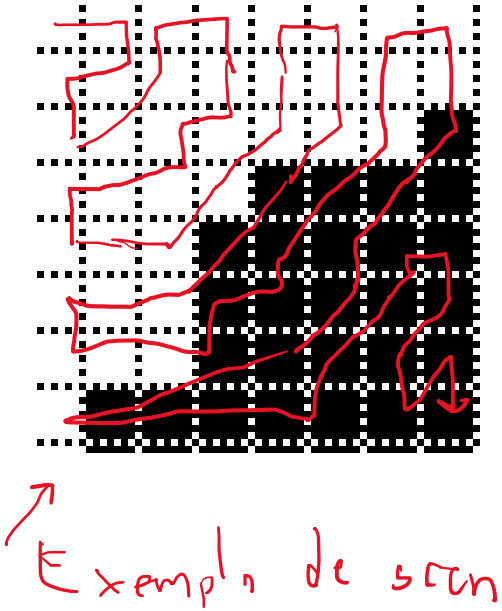


Plano 1

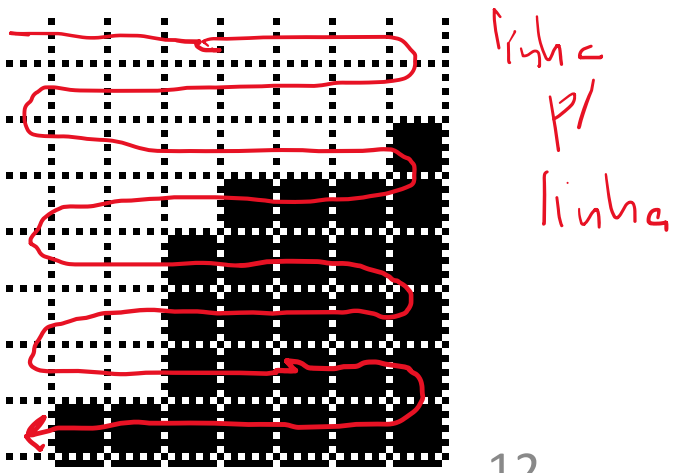
Scan

- A sequencia de scan é diferente para diferentes protocolos e, nem sempre segue as linhas normais. O scan busca maximizar corridas se elas são utilizadas posteriormente (RLE)





O formato exato do scan é definido pelo protocolo.
Nos exercícios adiante usamos um scan simples:

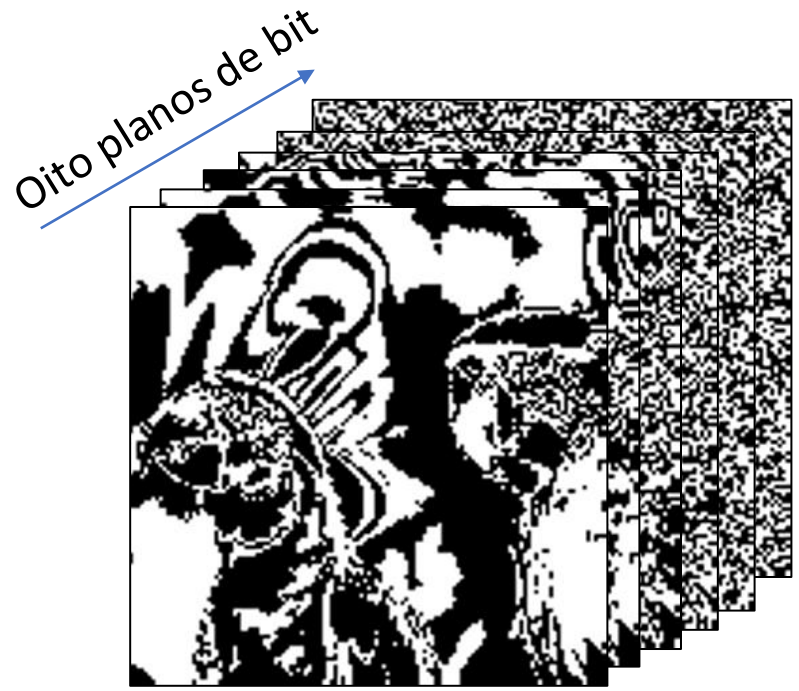


Scan

- Dentro de cada bloco, a ordem de leitura dos coeficientes não necessariamente segue a ordem de varredura usual da página
- A razão disso é que se deseja maximizar corridas e facilitar o encontro de redundâncias
  - A leitura usual linha a linha da esquerda para a direita gera interrupções de continuidade quando se lida com objetos
- O método (ordem) de leitura de um bloco é chamado scan, e, em geral fixo e definido pelo protocolo de compressão da imagem.

Exemplo de bitplanes

Imagem em tons de cinza (8 bits p/ pixel)



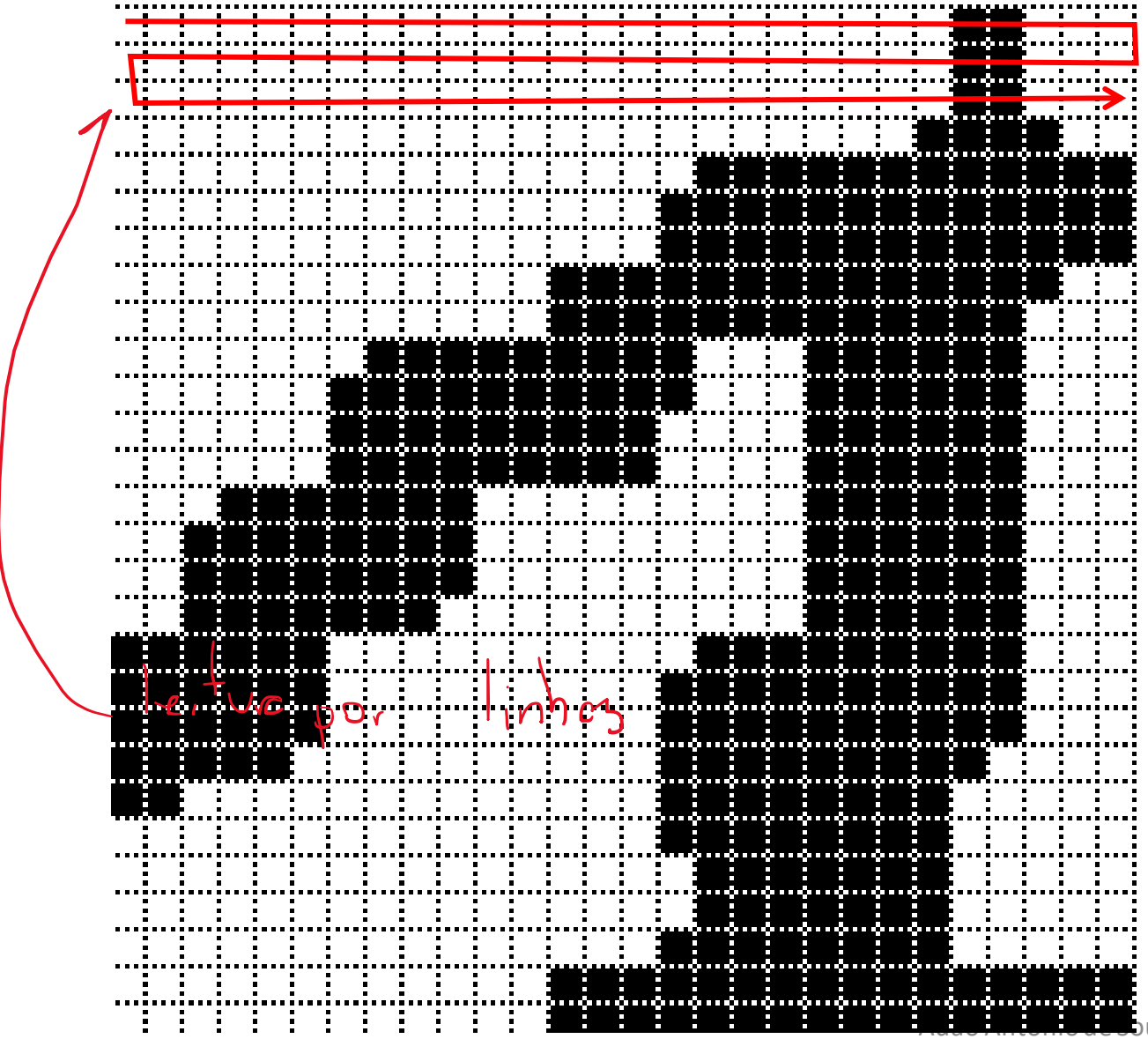
Plano 8

Plano 7

Codificação dos coeficientes

- A partir dos dados lidos em cada bloco se pode codificar os coeficientes
  - É muito comum fazer isso em corridas, assim as tuplas irão formar o alfabeto final.
  - Em alguns casos se pode optar por codificar os blocos como alfabeto (mais útil em casos monocromáticos, porém pouco usual)

Se a imagem toda fosse essa (28 x 28):



Poderiamos fazer scan tentando maximizar corridas:
p/Ex: Alterna o sentido de leitura a cada linha, esquerda para direit, volta para esquerda na próxima linha para a direita na seguinte e assim vai...

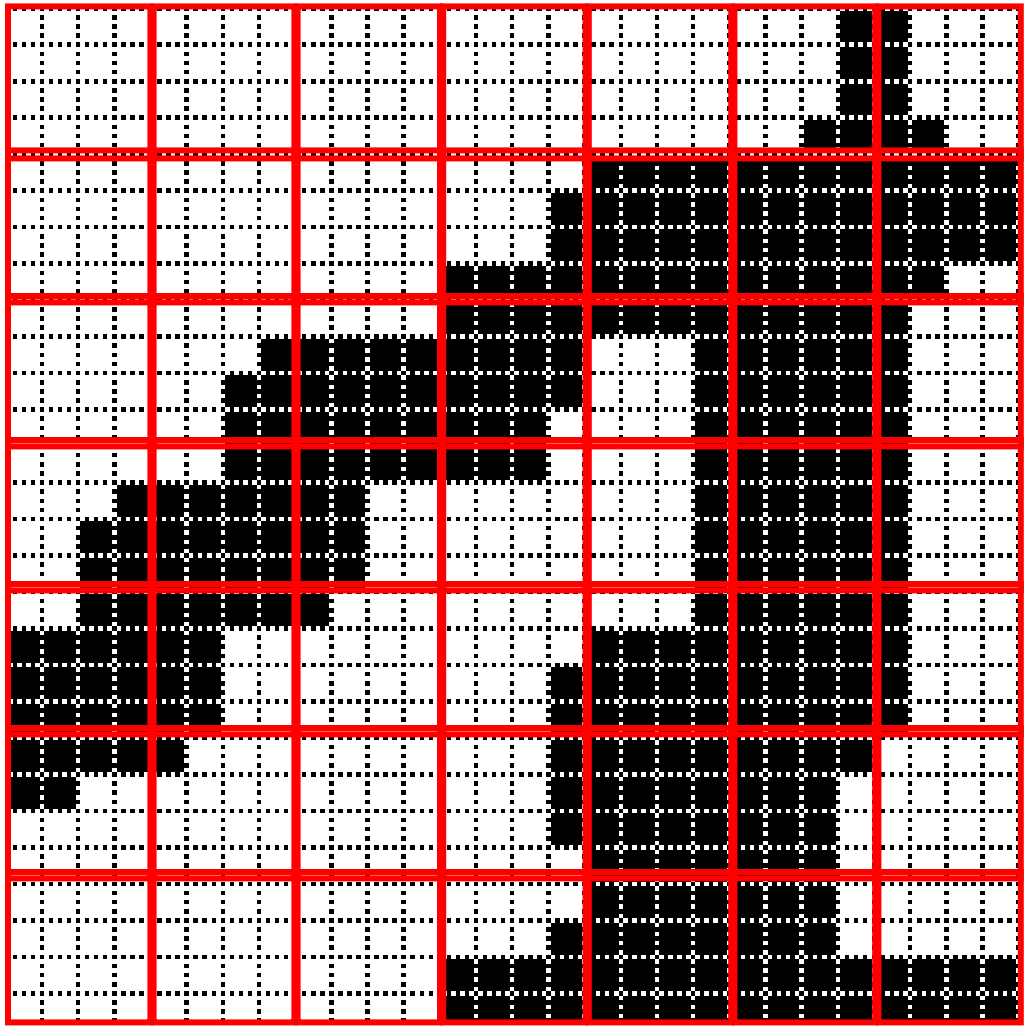
Nesse caso, as tuplas seriam:
(1,23)(0,2)(1, 26)(0,2)(1,26)(1,25)(0,4)(1,18)(0,12)(1,17)
(0,13)(1,17)(0,13)(1,12)(0, ...

Nessa situação:
1 – Quais seriam as próximas 4 tuplas?

2 – Quais serão as ultimas 4 tuplas da imagem?

3 – Nesse caso, se eu souber se o primeiro ponto da imagem é branco ou preto, é necessário guardar as duas informações de cada tupla?

Bitplanes sem RLE



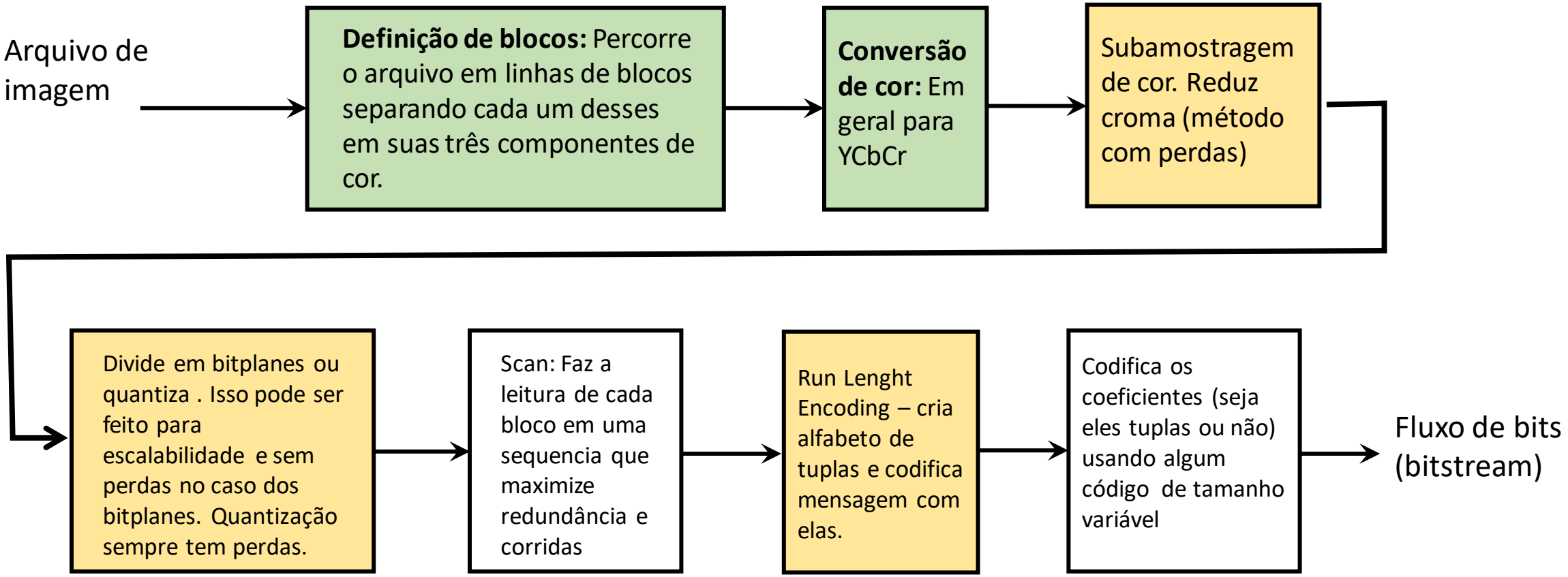
- Ao invés de codificar os bitplanes como corrida é possível codifica-los como blocos de bits.
- Nesse exemplo dividimos toda a imagem em blocos de 4x4
- Pode-se mapear cada bit de um bloco 4x4 para um numero inteiro de 16 bits, por exemplo como abaixo:

$$\begin{bmatrix} C_0 & C_1 & C_2 & C_3 \\ C_4 & C_5 & C_6 & C_7 \\ C_8 & C_9 & C_{10} & C_{11} \\ C_{12} & C_{13} & C_{14} & C_{15} \end{bmatrix} \rightarrow [C_{15}\ C_{14}\ C_{13}\ C_{12}\ C_{11}\ C_{10}\ C_9\ C_8\ C_7\ C_6\ C_5\ C_4\ C_3\ C_2\ C_1\ C_0]$$

- A sequencia, iniciando da primeira linha seria: FFFF, FFFF, FFFF, FFFF, FFFF, 3777,CEEE, FFFF, FFFF, FFFF, 077F, 0000, 0000, C000, 0000, ...
- Quais seriam os próximos quatro valores?
- Quais seriam os últimos quatro valores?

Fluxo de codificação em uma imagem

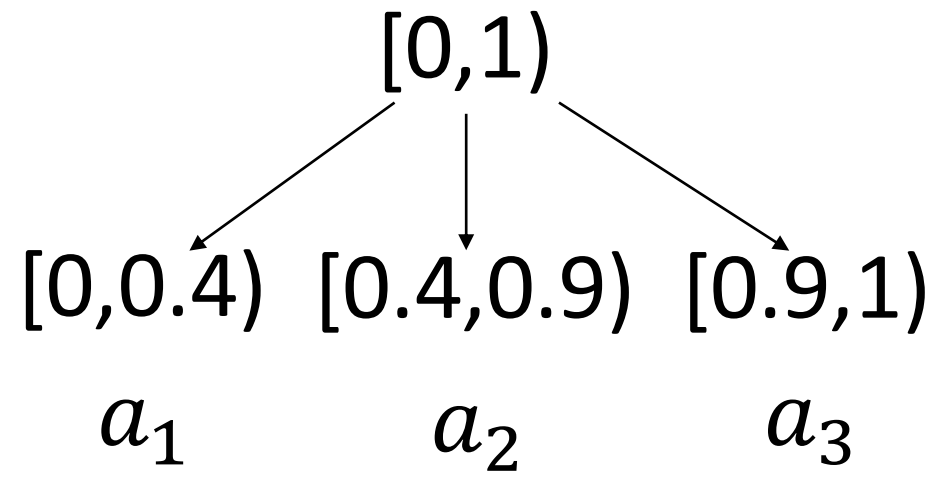
A codificação da imagem (típica) demanda diversas etapas algumas das quais são opcionais (laranja)



Codificação aritmética

- A codificação aritmética busca representar uma mensagem inteira como um único número real.
  - Esse número é então transformado em uma sequencia binária
- O conceito fundamental é que cada elemento do conjunto de símbolos da mensagem é visto como um elemento para estreitar o intervalo unitário.
- Cada mensagem corresponde portanto a um número único no intervalo de [0, 1)

Exemplo: Nosso conjunto é dado por $\Xi = \{a_1, a_2, a_3\}$, com probabilidades $p_1 = 0.4, p_2 = 0.4, p_3 = 0.1$. Cada símbolo do alfabeto corresponde a um sub-intervalo.



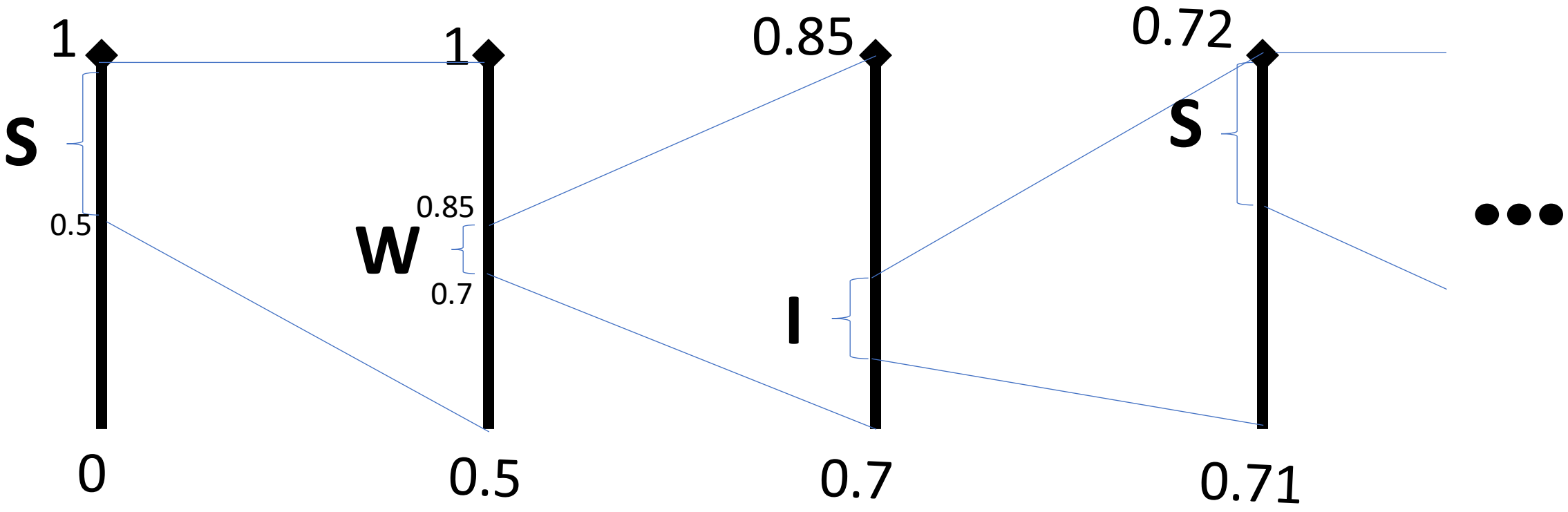
Como eu codifico uma mensagem?

Ex: $m = a_2 a_2 a_2 a_3$ ⟶ [0.8125, 0.825)

Codificação

- Inicia no intervalo [0, 1)
- Seleciona o subintervalo correspondente ao primeiro símbolo da mensagem.
  - Esse novo intervalo é agora o intervalo total
  - Seleciona o subintervalo desse novo intervalo que corresponde ao segundo símbolo
  - Repete até o último símbolo
- Ao final da codificação os números que representam os limites do intervalo carregam toda a mensagem
  - Pode-se utilizar o limite inferior ou superior de acordo com a conveniência

- A codificação funciona como um processo de “zoom” em que a cada passo selecionamos a parte do segmento que corresponde ao novo símbolo



Decodificação

- A partir do limite dado pelo valor, busca encaixar ele nas faixas do alfabeto e determina o primeiro símbolo
  - Ex: 0.71753375, fica no intervalo [0.5, 1) logo o primeiro símbolo é S
- Remove o efeito do símbolo decodificado
  - Ex: (0.71753375 – 0.5)/(1-0.5) = 0.4350675
- Localiza o novo limite nas faixas do alfabeto e extrai o novo símbolo
  - Ex: 0.4350675, fica no intervalo [0.4,0.5) logo o segundo símbolo é W
  - (0.4350675-0.4)/(0.5-0.4) = 0.350675
- Segue nesse processo até chegar a 0 (ou 1 se usar limite superior)

Gerando a sequencia de bits

- Até o momento conseguimos atingir parte do objetivo: temos um único número que representa toda a mensagem.
- No entanto, esse é um número real e, para fins de codificação precisamos de um trem de bits de alguma forma.
  - Notem que a representação de números em ponto flutuante não é adequada para resolver nosso problema pois é limitada em mantissa
- O ideal é que representemos o número como um numero binário inteiro.
  - Ex: 0.99462270125 ~ 99462270125/2^36 = 0.1111 1110 1001 1111 1001 0111 1110 0101 1011

Métodos de Dicionário

Prof. Adão Antonio de Souza Junior - IFSul

Compressão de strings

- A compressão de um código para strings de símbolos, em média, é melhor que para símbolos individuais.
- P/Ex: Imagine dois símbolos a_1 e a_2 com probabilidades 0.8 e 0.2
- Combinando de dois e de três:

String	Probability	Code
a_1a_1	$0.8 \times 0.8 = 0.64$	0
a_1a_2	$0.8 \times 0.2 = 0.16$	11
a_2a_1	$0.2 \times 0.8 = 0.16$	100
a_2a_2	$0.2 \times 0.2 = 0.04$	101

String	Probability	Code
$a_1a_1a_1$	$0.8 \times 0.8 \times 0.8 = 0.512$	0
$a_1a_1a_2$	$0.8 \times 0.8 \times 0.2 = 0.128$	100
$a_1a_2a_1$	$0.8 \times 0.2 \times 0.8 = 0.128$	101
$a_1a_2a_2$	$0.8 \times 0.2 \times 0.2 = 0.032$	11100
$a_2a_1a_1$	$0.2 \times 0.8 \times 0.8 = 0.128$	110
$a_2a_1a_2$	$0.2 \times 0.8 \times 0.2 = 0.032$	11101
$a_2a_2a_1$	$0.2 \times 0.2 \times 0.8 = 0.032$	11110
$a_2a_2a_2$	$0.2 \times 0.2 \times 0.2 = 0.008$	11111

Compressão de strings

- Se calcularmos o tamanho médio por símbolo, no caso de 1 bit teremos: 0.8x1+0.2x1 = 1bit/símbolo

- No caso de dois a dois teremos: 0.61x 1 + 0.16x 2 + 0.16x 3 + 0.04x3 = 0.78 bits/símbolo

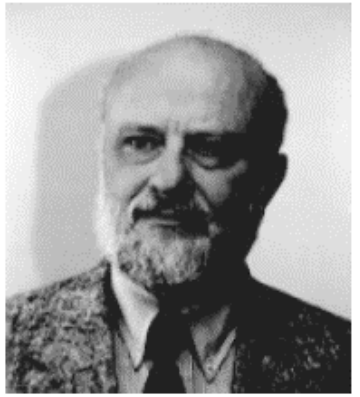
- No caso três a três teremos: 0.512x1+0.128x3+0.128x3+0.128x0.032x5+

+0.032x5+0.032x5+0.008x5=0.728 bits/simbolo

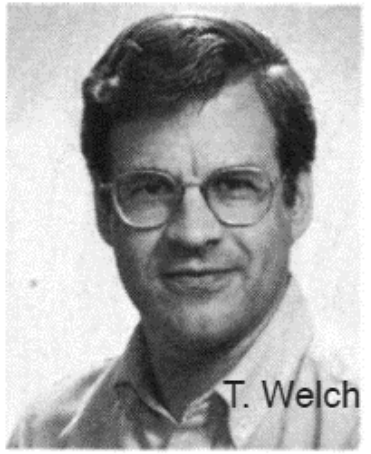
qto + símbolos melhor a média.

Os métodos Lempel-Ziv


J. Ziv


A. Lempel

The LZW Trio. Having one's name attached to a scientific discovery, technique, or phenomenon is considered a special honor in science. Having one's name associated with an entire field of science is even more so. This is what happened to Jacob Ziv and Abraham Lempel. In the late 1970s these two researchers developed the first methods, LZ77 and LZ78, for dictionary-based compression. Their ideas have been a source of inspiration to many researchers who generalized, improved, and combined them with RLE and statistical methods to form many popular lossless compression methods for text, images, and audio. Close to 30 such methods are described in this chapter, some in great detail. Of special interest is the popular LZW algorithm, partly devised by Terry Welch (Section 6.13), which has extended LZ78 and made it practical and popular.


T. Welch

Lempel-Ziv

- Os dois métodos pioneiros utilizam algoritmos para construir um dicionário a medida em que se codifica a mensagem
- Ao contrário de RLE, que apenas procura padrões de repetição simples (n vezes tal caractere) esses métodos permitem encontrar combinações de símbolos que se repetem ao longo do texto.
- A codificação final utiliza esse dicionário, seguido por algum método de tamanho variável para gerar o trem de bits.
- Iremos apresentar os métodos LZ77 e LZ78, outros algoritmos podem ser encontrados na literatura suplementar.

LZ77

- O método utiliza duas janelas ou buffers:
  - Search buffer: tipicamente com milhares de bytes de comprimento
  - Look-ahead buffer: tipicamente com algumas dezenas de bytes de comprimento:

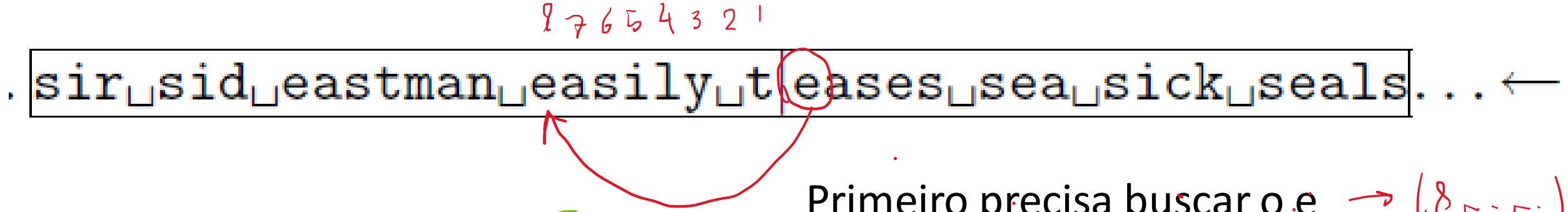
- O método recebe o texto em bytes como entrada e gera tokens como saída (d,l,s)
- Os tokens formam o novo alfabeto que será usado para gerar a bitstream

Codificação LZ77

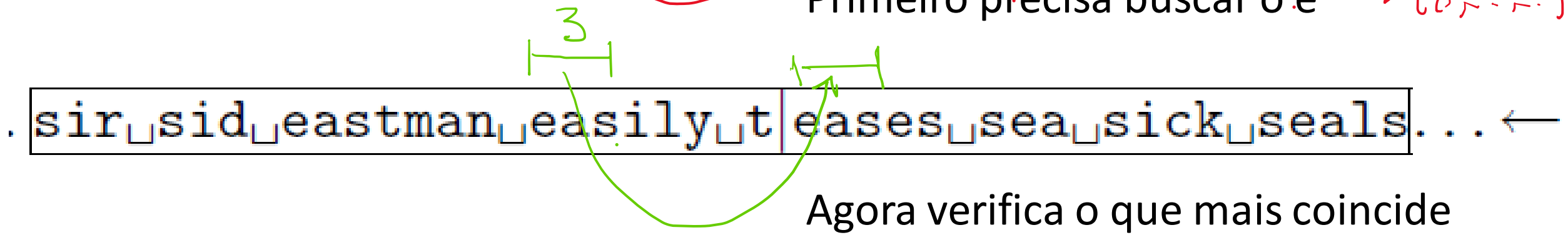
- P. ex: Imaginem a seguinte mensagem “sir_sid_eastman_easily_teases_sea_sick_seals”
- Inicialmente vamos pensar como se as primeiras quatro palavras “sir_sid_eastman_easily_t” já houvessem sido codificadas e, portanto já estivessem no *search buffer* e agora estivéssemos codificando os próximos símbolos.
  - Depois falamos sobre como o processo começa

Algoritmo

- Para cada símbolo no LAB (look ahead buffer) faz uma busca de trás para frente no SB (Search buffer) procurando uma coincidência.
  - Quando encontra indica a distância em que encontrou (d,...,...)
    - A partir daquele ponto verifica se o próximo símbolo do LAB coincide com o próximo do SB após o encontrado.
      - Caso afirmativo segue fazendo isso até que não seja mais verdade e anota quantos símbolos coincidiram l → (d, l , ...)
    - Anota o símbolo que não foi achado na sequencia → (d, l, s)
    - Continua procurando outros matches que gerem l maior
  - Quando não encontra nada anota o símbolo antecedido por dois zeros → (0,0,s)
  - Mais fácil entender no exemplo:

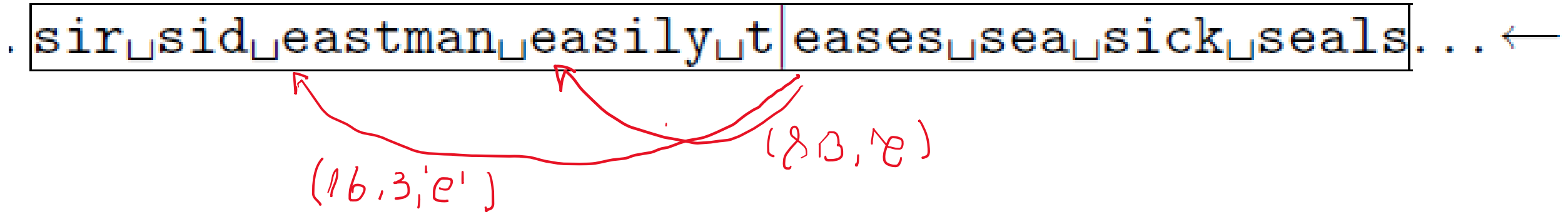


Primeiro precisa buscar o e → (8,...)



Agora verifica o que mais coincide
'eas' é igual, logo → (8,3,'e')

Será que tem melhores opções??? Segue procurando até acabar o buffer...



Algoritmo (cont.)

- No exemplo dado encontrou dois possíveis tokens:
- (8,3,’e’) e (16,3,’e’)
- Se o comprimento de algum fosse o maior escolheria esse.
  - Não é o caso, os dois tem comprimento (l=3)
  - Se a melhor opção de comprimento tiver mais de um possível valor, escolhe o último encontrado (mais distante)
  - Assim a codificação ficaria (16,3,’e’)
  - Isso, na época, simplificava o uso de memória pelo algoritmo, se, no entanto usarmos algo como huffmann depois, a melhor escolha é escolher o primeiro encontrado (mais próximo). Isso é um dos parâmetros do algoritmo.

Inicializando...

- No início do algoritmo é simples perceber que muitos tokens terão distância e comprimento zero, até o preenchimento do SB

	sir␣sid␣eastman␣	⇒	(0,0,“s”)
s	ir␣sid␣eastman␣e	⇒	(0,0,“i”)
si	r␣sid␣eastman␣ea	⇒	(0,0,“r”)
sir	␣sid␣eastman␣eas	⇒	(0,0,“␣”)
sir␣	sid␣eastman␣easi	⇒	(4,2,“d”)

- Note que usamos nesse exemplo um SB de 24 símbolos e um LAB de 16 símbolos e complete a codificação da frase...

	sir␣sid␣eastman␣	$\Rightarrow$	$(0,0,\text{“s”})$
s	ir␣sid␣eastman␣e	$\Rightarrow$	$(0,0,\text{“i”})$
si	r␣sid␣eastman␣ea	$\Rightarrow$	$(0,0,\text{“r”})$
sir	␣sid␣eastman␣eas	$\Rightarrow$	$(0,0,\text{“␣”})$
sir␣	sid␣eastman␣easi	$\Rightarrow$	$(4,2,\text{“d”})$

sir_sid_eastman_easil ⇒ (4,1,_)

sir_sid_eastman_easily ⇒ (0,0,'e')

sir_sid_eastman_easily_ ⇒ (0,0,'a')

sir_sid_eastman_easily_te ⇒ (10,2,'m')

sir_sid_eastman_easily_tease

LZ78

- Não possui SB, LAB ou janela deslizante
- Tem um dicionário de combinações que começa vazio (NULL)
- O token tem dois campos (p, s)
  - p é um ponteiro para o dicionário
  - s é um novo símbolo
- O token não armazena informação de comprimento pois essa fica implicita
- Novamente o melhor para explicar é um exemplo.
  - Novamente a mensagem é: “sir_sid_eastman_easily_teases_sea_sick_seals”

Primeiros quatorze passos de codificação

Dictionary		Token	Dictionary		Token
0	null				
1	s	(0,s)	8	a	(0,a)
2	i	(0,i)	9	st	(1,t)
3	r	(0,r)	10	m	(0,m)
4	␣	(0,␣)	11	an	(8,n)
5	si	(1,i)	12	␣ea	(7,a)
6	d	(0,d)	13	sil	(5,l)
7	␣e	(4,e)	14	y	(0,y)

Quais seriam os próximos passos???

sir_sid_eastman_easily_teases_sea_sick_seals”

15 '_t' (9, t)

16 'e' (0, e)

17 'as' (8, s)

18 'es' (16, s)

Revisão

- Observe o código de LZ77 e LZ78
- Finalize a codificação da frase usando LZ77 (manualmente) e LZ(78) manualmente
- Utilize o MATLAB para verificar as respostas
- Caso utilizemos LZ77 para codificar o texto fornecido, qual será a sequencia de tokens de saída?
  - Calcule a probabilidade de cada token
  - Avalie a entropia do conjunto de tokens
  - Calcule o tamanho mínimo da bitstream gerada por essa codificação
- Repita o procedimento para LZ78
- Façam as duas leituras adicionais pois elas agregam muito ao explicar os diversos métodos derivados, em especial LZR

Isso tem implicações mais adiante...

A codificação da imagem (típica) demanda diversas etapas algumas das quais são opcionais (laranja)