# Video Compression
# 視訊壓縮

Yu-Lun (Alex) Liu
劉育綸

with slides by Wen-Hsiao Peng,
Shao-Yi Chien,
Hsueh-Ming Hang,
and Aggelos K. Katsaggelos

| Week | Date | Topic | Assignments |
|---|---|---|---|
| 1 | 2025-09-01 | | |
| 2 | 2025-09-08 | Introduction to Image and Video Processing | |
| 3 | 2025-09-16 | Signals and Systems | #1 – Color Transform, due: 2025-09-29 1:19pm |
| 4 | 2025-09-22 | Fourier Transform and Sampling | |
| 5 | 2025-09-29 | 教師節補假 | |
| 6 | 2025-10-06 | 中秋節 | |
| 7 | 2025-10-13 | Fourier Transform and Sampling | #2 – 2D-DCT, due: 2025-10-27 1:59pm |
| 8 | 2025-10-20 | Motion Estimation | Final project assigned (group together in fours) |
| 9 | 2025-10-27 | Lossless Compression | #3 – MEMC, due: 2025-11-10 1:59pm |
| 10 | 2025-11-03 | Image Compression | |
| 11 | 2025-11-10 | Video Compression | #4 – Entropy coding, due: 2025-11-24 1:59pm |
| 12 | 2025-11-17 | Learning-based Image/Video Compression | |
| 13 | 2025-11-24 | Paper Presentation | |
| 14 | 2025-12-01 | Guest Lecturer – MEDIATEK Meta NETFLIX | |
| 15 | 2025-12-08 | Guest Lecturer – MEDIATEK Meta NETFLIX | |
| 16 | 2025-12-15 | Final Project Presentation | |

# Lossless Compression

with slides by Wen-Hsiao Peng, Shao-Yi Chien, Hsueh-Ming Hang, and Aggelos K. Katsaggelos

# Lossless Compression

- Introduction
- Elements of Information Theory – Part I
- Elements of Information Theory – Part II
- Huffman Coding
- Run-Length Coding and Fax
- Arithmetic Coding
- Dictionary Techniques
- Predictive Coding

# Lossless Compression

- **Introduction**
- Elements of Information Theory – Part I
- Elements of Information Theory – Part II
- Huffman Coding
- Run-Length Coding and Fax
- Arithmetic Coding
- Dictionary Techniques
- Predictive Coding

# Definition

- What is Compression?
  - The minimization of the number of bits in representing an image or video, while
    - Being able to exactly reconstruct the original data (**lossless compression**)

    or
    - Maintaining an acceptable quality of the reconstructed original data (**lossy compression**)

# Why Compress?

| | |
|---|---|
| Telephony (220-3,400 Hz) | 8,000 samples/s x 12 b/sample = 96 kbps |
| Wideband speech (50-7,000 Hz) | 16,000 samples/s x 14 b/sample = 224 kbps |
| Wideband audio (20-20,000 Hz) | 44,100 samples/s x 2 channels x 16 b/sample = 1.412 Mbps |
| Color images | 512 x 512 pixels x 24 bpp = 6.3 Mbits = 786 Kbytes |
| Video (CCIR601) | 720 x 480 gray pixels x 8 bpp x 30 frames/s +<br>2 x 360 x 480 chroma pixels x 8 bpp x 30 frames/s = 166 Mbps |
| HDTV | 1280 x 720 color pixels x 24 bpp x 60 frames/s = 1.3 Gbps |

CCIR601: Without compression we could store

- 31 s of videos on a CD-ROM (650 Mbytes)
- 4 minutes of video on a DVD-5 (40 Gbits)

HDTV Terrestrial Broadcasting:

- 19.3 Mbits/s (6 MHz channel)
- Need compression ratio ~70

# Why are Signals Compressible?

- Statistical redundancy or structure in the data (spatial, temporal, spectral)
- Existence of perceptually irrelevant information

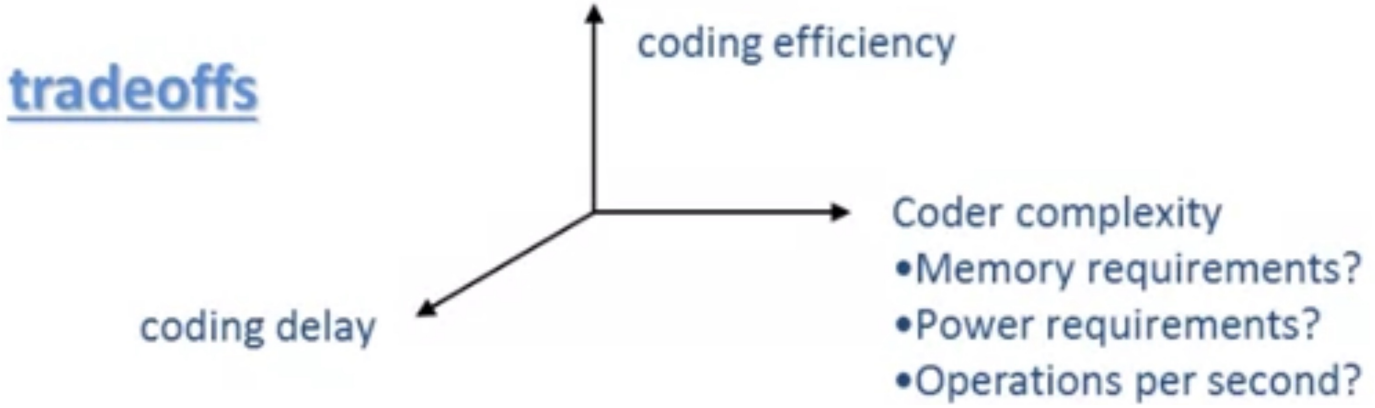
Compression is one of the **enabling technologies** behind the multimedia revolution we are experiencing

# Proliferation of Compression Application

- Over 50 years of extraordinary theoretical results on compression
- Use of perceptually based distortion measures
- Successful standards
- Unparalleled DSP capabilities
- Technological advances in computer, networks, and telecommunications

# Lossless Compression

- **OBJECTIVE**: Exact reconstruction of original data from compressed data, i.e., a **reversible** process
- **Compression ratio** (input size over output size) is determined by the **entropy** of the source
- Low Compression ratios
- Application examples: text, medical images, **lossy codecs**



# Lossless Source Coding

**Statistical Methods (source statistics known)**

- Huffman
- Extended Huffman
- Other (Gilbert, Fano)

Applications

- Group 3 FAX
- Group 4 FAX

**Universal Methods (source statistics unknown)**

- Arithmetic Coding
- Dictionary Techniques (LZxx, LZW)
- Other (adaptive Huffman)

Applications

- JBIG (arithmetic)
- Unix Compress, GIF, V.42 bis (LZW)

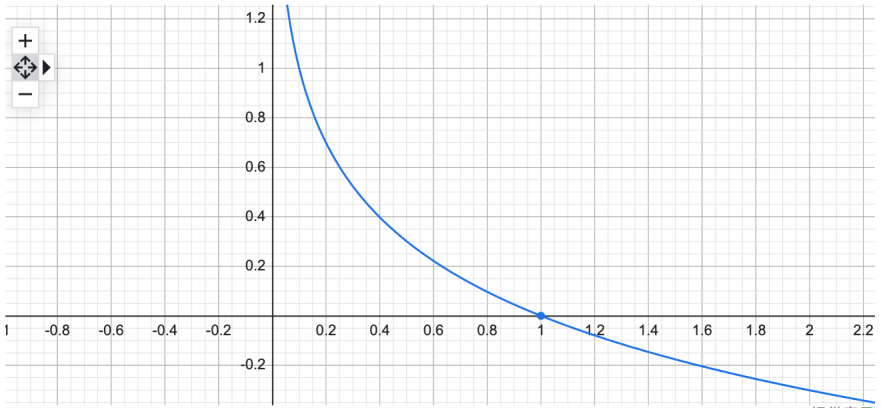
# Lossless Compression

- Introduction
- **Elements of Information Theory – Part I**
- Elements of Information Theory – Part II
- Huffman Coding
- Run-Length Coding and Fax
- Arithmetic Coding
- Dictionary Techniques
- Predictive Coding

# Elements from Information Theory

- Any information generating process can be viewed as a source that emits a sequence of symbols randomly chosen from a finite alphabet
  - Example: natural written language; $n$-bit images ($2^n$ symbols)
- Simplest case: discrete memoryless source (DMS) – successive symbols produced by the source are statistically independent (and identically distributed)
- A DMS is completely specified by the source alphabet $S = \{s_1, s_2, \ldots, s_n\}$ and the associated probabilities $\{p_1, p_2, \ldots, p_n\}$

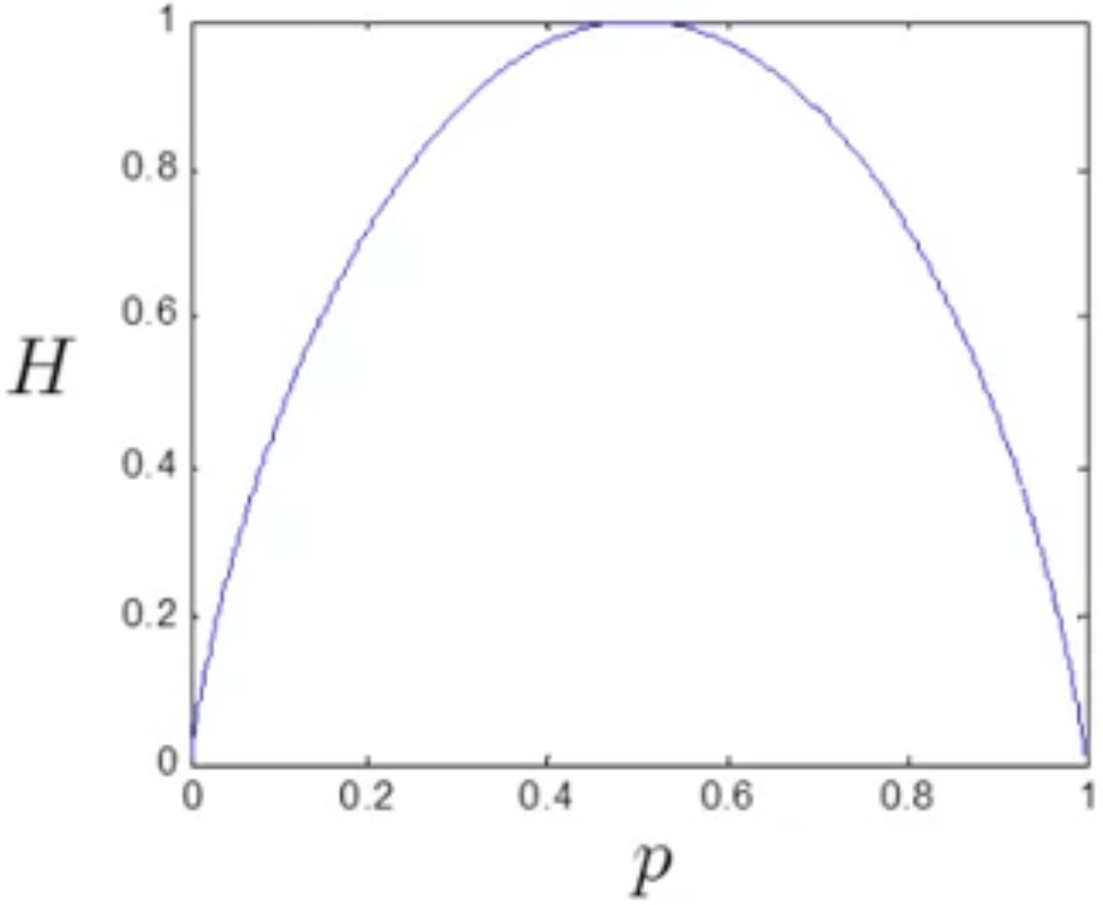
# Entropy




- Self Information $I(s_i) = \log \frac{1}{p_i} = -\log p_i$
  - Base of log: 2 -> unit: bits; e -> nats; 10 -> Hartleys
  - The occurrence of a less probable event provides more information
  - The information of independent events taken as a single equals the sum of the information
  - $I(s_1 s_2) = \log \frac{1}{p(s_1 s_2)} = \log \frac{1}{p(s_1)p(s_2)} = -\log\big(p(s_1)p(s_2)\big) = -\log p_1 - \log p_2 = I(s_1) + I(s_2)$
- Entropy of a DMS: Average information per symbol

$$H(S) = \sum_{i=1}^{n} p_i I(s_i) = -\sum_{i=1}^{n} p_i \log_2 p_i$$

# Plot of Entropy




$n$ symbols

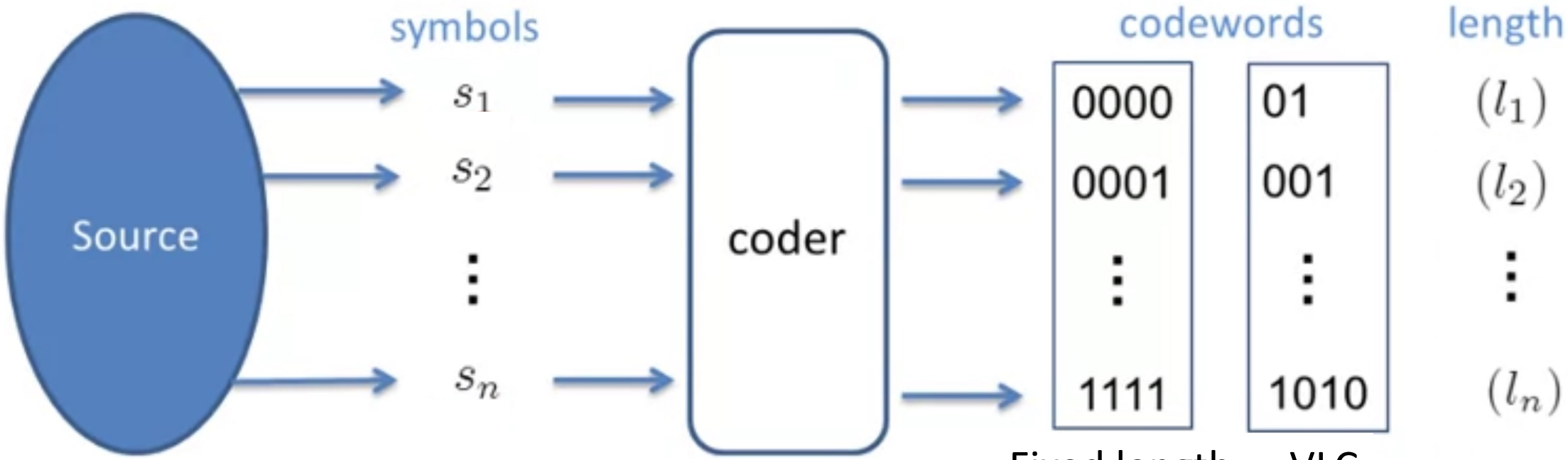
$H$ is maximum

When $p_i = 1/n$

$H_{\max} = \log_2 n$

$$H = -p \log_2 p - (1-p) \log_2 (1-p)$$

# Coding




Fixed length VLC

Alphabet: $A = \{s_1, s_2, \cdots, s_n\}$

$p_1$ $p_2$ $p_n$

Average Codeword Length: $l_{avg} = \sum_{i=1}^{n} l_i p_i$

ASCII
a(1000011)
A(1000001)

Morse principle
e(·)
a(· –)
q(– – · –)

# Coding

- **First order coders**: encode one symbol at a time, independently of other symbols
- **Block codes**: Group source into blocks of symbols; each block $N$ considered as a single source symbol generated by a source $S_N$ with alphabet size $n^N$.

  In this case: $H(S_N) = N \times H(S)$
- **Non-block codes**: arithmetic

# Source Coding Theorem

- Let $S$ be a source with alphabet size $n$ and entropy $H(S)$. Consider coding blocks of $N$ source symbols into binary codewords. For and $\delta > 0$, it is possible by choosing $N$ large enough to construct a code in such a way that the average number of bits per original source symbol $l_{\mathrm{avg}}$ satisfies

$$H(S) \leq l_{\mathrm{avg}} < H(S) + \delta$$

# Lossless Compression

- Introduction
- Elements of Information Theory – Part I
- **Elements of Information Theory – Part II**
- Huffman Coding
- Run-Length Coding and Fax
- Arithmetic Coding
- Dictionary Techniques
- Predictive Coding

# Entropy of a Source

- Entropy of the English language
  - DMS, equal probabilities: H(S) = 4.70 bits / letter (4.76 bits / letter for 27 letters)
  - DMS: H(S) = 4.14 bits / letter
  - Taking co-dependencies up to 8 letters: H(S) = 2.3 bits / letter
- Knowledge about the structure of the data

  Example #1: consider data: 1234323456
  - P(2) = P(4) = 0.2, P(3) = 0.3, P(1) = P(5) = P(6) = 0.1 -> H = 2.44 bits / s
  - Predictor: $x_n = x_{n-1} + r_n$
  - $r_n$ = 1111-1-11111: P(1) = 0.8, P(-1) = 0.2 -> H = 0.7 bits / s

  Example #2: consider data: 334533334545
  - P(3) = ½, P(4) = P(5) = ¼ -> H = 1.5 bits / symbol
  - P(33) = P(45) = ½ -> H = 1 bit / new symbol -> 0.5 bits / original symbol

# Example

| Symbols | Probability | Code 1 | Code 2 | Code 3 | Code 4 |
|---|---|---|---|---|---|
| $s_1$ | 1/2 | 0 | 0 | 0 | 0 |
| $s_2$ | 1/4 | 0 | 1 | 10 | 01 |
| $s_3$ | 1/8 | 1 | 00 | 110 | 011 |
| $s_4$ | 1/8 | 10 | 11 | 111 | 0111 |
| Average length | | 1.125 | 1.25 | 1.75 | 1.875 |

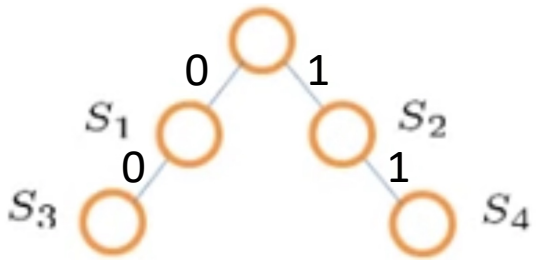
010110

Entropy H(S) = 1.75 bits / symbol

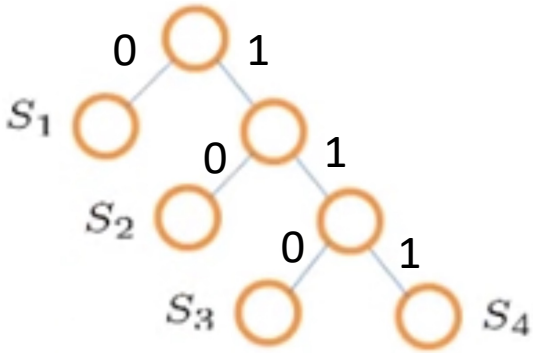
- **Uniquely decodable (UD)** or uniquely decipherable: any finite sequence of codewords corresponds to only one message sequence
- **Prefix**(-free) **code**: no codeword is prefix to another

# Binary Tree Representation

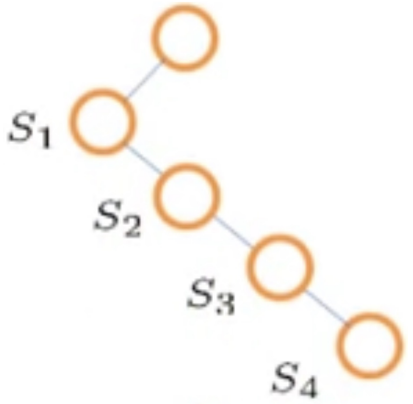
| Code 2 |
|---|
| 0 |
| 1 |
| 00 |
| 11 |




| Code 3 |
|---|
| 0 |
| 10 |
| 110 |
| 111 |




| Code 4 |
|---|
| 0 |
| 01 |
| 011 |
| 0111 |




- Base of the tree: **root node**
- Point at which a branch divides: **branch node**
- Termination point: **leaf (or external) node**
- In a prefix code, codewords are **only** associated with leaves

# Prefix and UD Codes

- In general, it is hard to tell if a code is UD (systematic procedure exists to determine if a code is UD in finite number of steps)
- Prefix codes are UD (the converse is not true)
- Prefix codes are instantaneously decodable
- Prefix codes: easy to design, easy to decode
- No loss of performance: For any UD code that is not prefix, we can find a prefix with the same rate.

# Lossless Compression

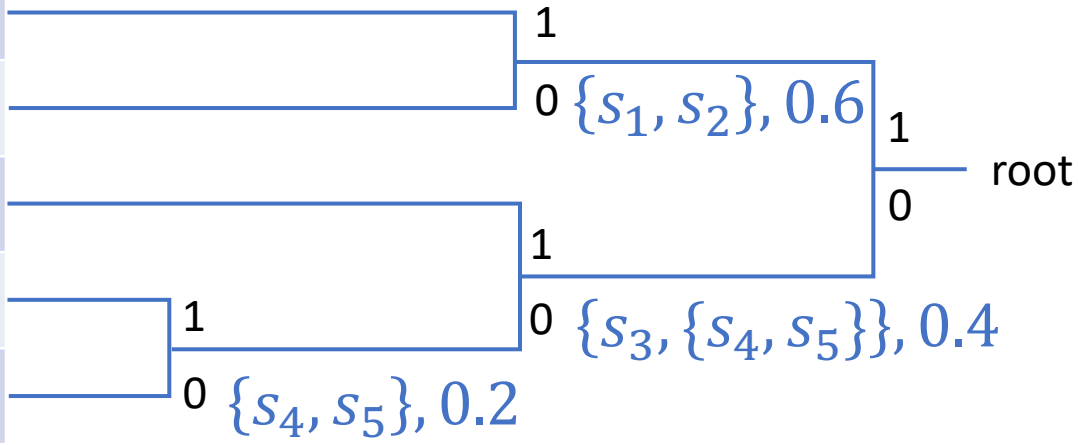
- Introduction
- Elements of Information Theory – Part I
- Elements of Information Theory – Part II
- **Huffman Coding**
- Run-Length Coding and Fax
- Arithmetic Coding
- Dictionary Techniques
- Predictive Coding

# Huffman Coding Algorithm (1951)

- Symbols that occur more often will have shorter codewords
- The two symbols that occur least frequently will have the same length
- Design procedure:
  - Sort the symbols according to probability
  - Combine the lease probable symbols to form a composite symbol, with probability equal to the sum of the respective symbol probabilities (with this step the size of the alphabet is reduced by one).
  - Repeat procedure for remaining symbols
  - Extract the codewords from the resulting binary tree representation of the code

# Huffman Code Example

| Symbol | Prob |
|---|---|
| $s_1$ | 0.2 |
| $s_2$ | 0.4 |
| $s_3$ | 0.2 |
| $s_4$ | 0.1 |
| $s_5$ | 0.1 |




$s_1$: 11
$s_2$: 10
$s_3$: 01
$s_4$: 001
$s_5$: 000

↑
Huffman code

$H \approx 2.122$ b/s
$l_{avg} \cong 2.2$ b/s

# Properties of Huffman Codes

$$H(S) \leq R < H(S) + 1$$

$$\begin{cases} p_{\max} < 0.5 \Rightarrow R < H(S) + p_{\max} \\ p_{\max} \geq 0.5 \Rightarrow R < H(S) + p_{\max} + 0.086 \end{cases}$$

- Best possible code when $p_i = 2^{-i}$: in such a case: $R = H(S)$
- If alphabet size is large, $p_{\max}$ is generally small, and Huffman codes perform quite well
- If alphabet size is small and $p_{\max}$ is large, the Huffman codes can be quite inefficient

# Huffman Code for Extended Source

- Block codes: Group source into blocks of symbols; each block $N$ considered as a single source symbol generated by a source $S_N$ with alphabet size $n^N$. Then $H(S_N) = N \cdot H(S)$
- Therefore

$$H(S_N) \leq R_N < H(S_N) + 1 \Rightarrow N \cdot H(S) \leq N \cdot R < N \cdot H(S) + 1$$

$$\Rightarrow H(S) \leq R < H(S) + \frac{1}{N}$$

- Tradeoff: blocking increases number of possible codewords (e.g., for $n = 3$ and $N = 8$, $3^8 = 6561$ codewords)
  - Storage, computation, delay

# Example of Extended Source: Case that Works

| Symbols | Probability | Code |
|---|---|---|
| $s_1$ | 0.8 | 0 |
| $s_2$ | 0.02 | 11 |
| $s_3$ | 0.18 | 10 |

Huffman code: entropy = 0.816 bits / symbol:
average length = 1.2 bits / symbol;
redundancy = 0.384 bits / symbol, or 47% of entropy

| Symbols | Probability | Code |
|---|---|---|
| $s_1s_1$ | 0.64 | 0 |
| $s_1s_2$ | 0.016 | 10101 |
| $s_1s_3$ | 0.144 | 11 |
| $s_2s_1$ | 0.016 | 101000 |
| $s_2s_2$ | 0.0004 | 10100101 |
| $s_2s_3$ | 0.0036 | 1010011 |
| $s_3s_1$ | 0.1440 | 100 |
| $s_3s_2$ | 0.0036 | 10100100 |
| $s_3s_3$ | 0.0324 | 1011 |

The extended alphabet and corresponding Huffman code;
average length = 1.7516 bits / new symbol;
or average length = 0.8758 bits / original symbol;
redundancy = 0.06 bits / symbol, or 7% of entropy

# Example of Extended Source: Case that does not Work

| Symbols | Probability | Code |
|---|---|---|
| $s_1$ | 0.95 | 0 |
| $s_2$ | 0.02 | 11 |
| $s_3$ | 0.03 | 10 |

Huffman code: entropy = 0.335 bits / symbol:
average length = 1.05 bits / symbol;
redundancy = 0.715 bits / symbol, or 213% of entropy

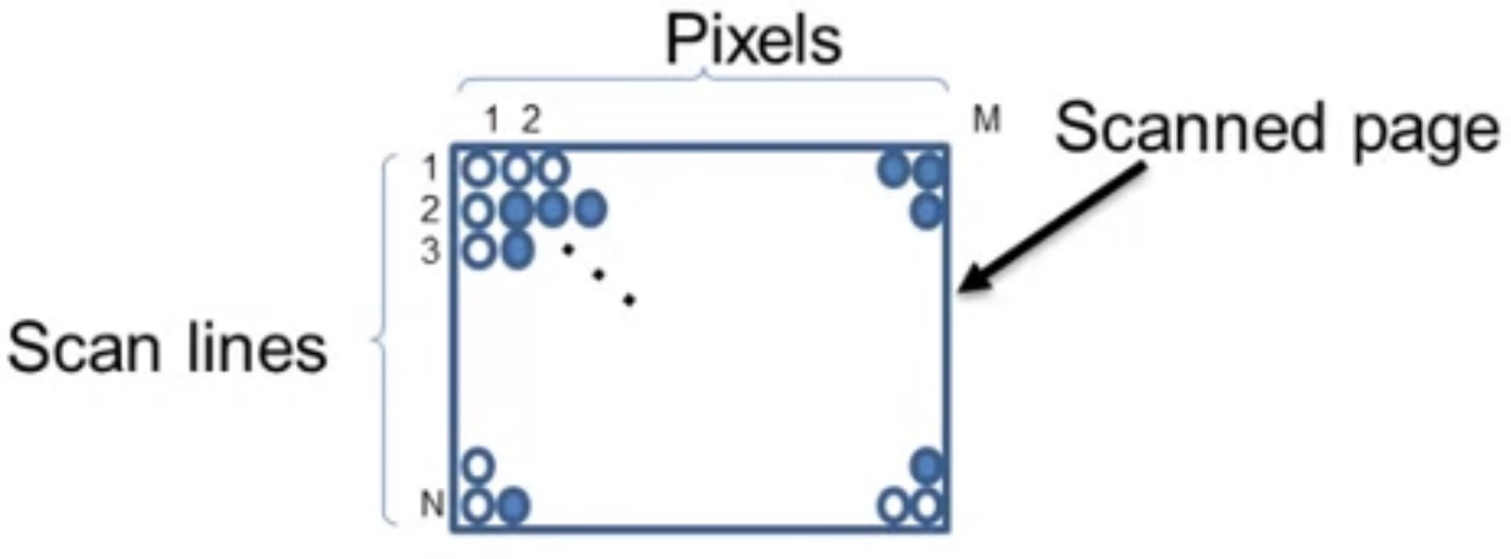
| Symbols | Probability | Code |
|---|---|---|
| $s_1s_1$ | 0.9025 | 0 |
| $s_1s_2$ | 0.0190 | 111 |
| $s_1s_3$ | 0.0285 | 100 |
| $s_2s_1$ | 0.0190 | 1101 |
| $s_2s_2$ | 0.0004 | 110011 |
| $s_2s_3$ | 0.0006 | 110001 |
| $s_3s_1$ | 0.0285 | 101 |
| $s_3s_2$ | 0.0006 | 110010 |
| $s_3s_3$ | 0.0009 | 110000 |

Huffman code for the extended alphabet;
average length = 1.222 bits / new symbol;
or average length = 0.611 bits / original symbol;
redundancy = 72% of entropy;
redundancy drops to acceptable values for N = 8 (alphabet size = 6561)

# Lossless Compression

- Introduction
- Elements of Information Theory – Part I
- Elements of Information Theory – Part II
- Huffman Coding
- **Run-Length Coding and Fax**
- Arithmetic Coding
- Dictionary Techniques
- Predictive Coding

# Facsimile Encoding

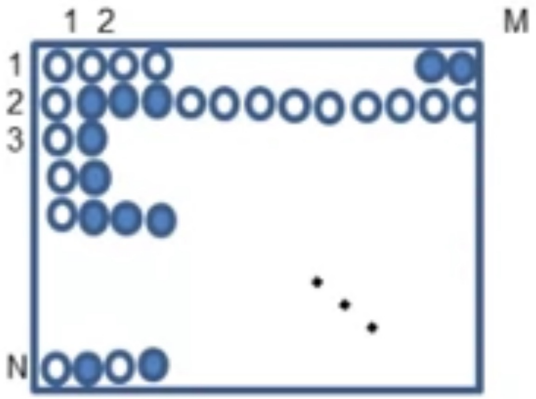



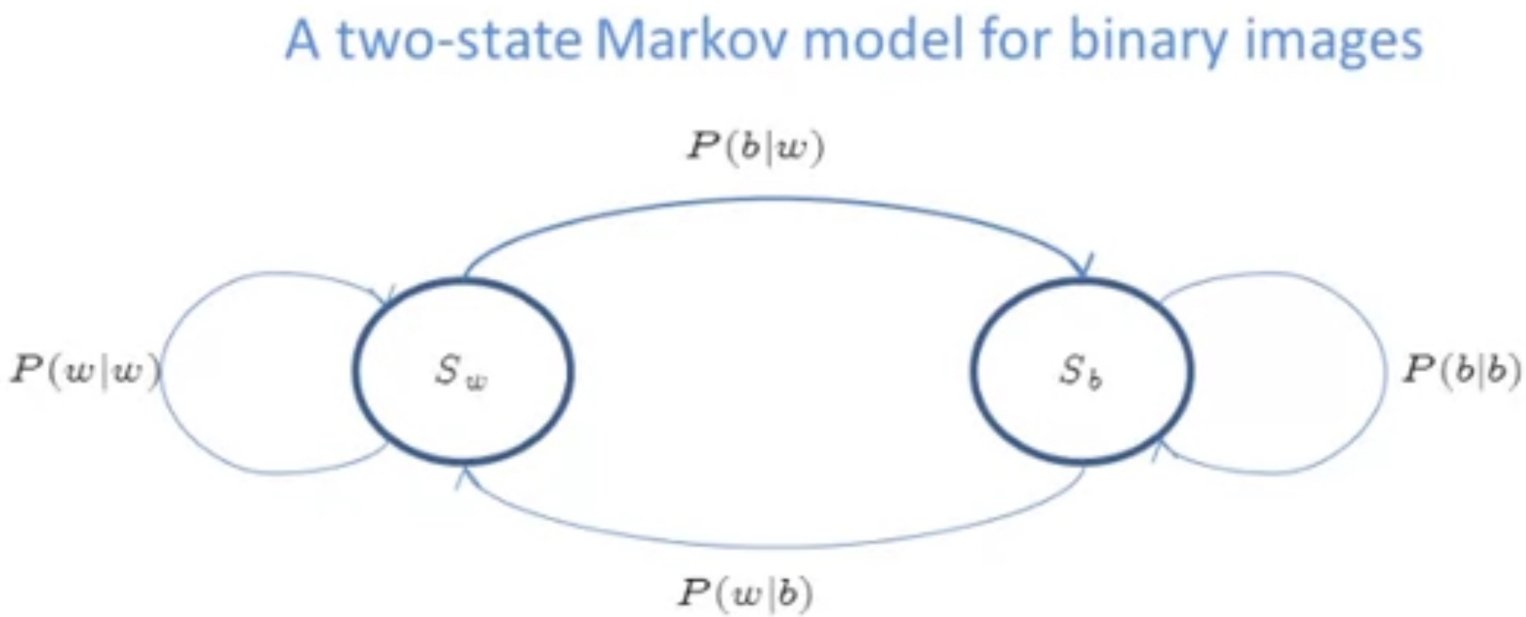

Resolution = M x N
Pixel resolution = 8 per mm
Line resolution = 3.85/7.7 per mm
= 100/200 per inch
A4 document: 210x297 mm

# G3, G4 Facsimile Standards

- Developed for text documents
  - Do not work as well for tables, charts, etc.
  - Efficiency drop dramatically for halftone images
- G3: digital operation with analog modems
  - **Run length coding** (1-D and 2-D)
  - **Huffman** compression for black and white runs (static Huffman tables)
  - 2-D: current line predicted from previous line
  - 1-D/2-D mixed to prevent error propagation
- G4: all-digital for digital networks (e.g., ISDN)
  - 2-D run length only (no error protection necessary)

# Run Length Coding




$$H = P(S_w)H(S_w) + P(S_b)H(S_b)$$

# G3 Facsimile Standards

- Designing 1-D Huffman codes
  - Estimate probabilities of runs of white pixels and runs of black pixels
  - Follow Huffman encoding procedure
- 1-D scheme (Modified Huffman – MH):
  - Must limit the number of possible lengths
  - $r_l = 64m + t, t = 0, 1, \dots, 63, m = 1, 2, \dots, 27$
  - Use codes for m (*make-up codes*) and t (*terminating codes*)
  - Maximum length 1728 (A4-size document)
  - Optional codes for wider documents

# ITU-T G3, G4 Fax Huffman Codes

Terminating codes →

| White run-length | Code-word | Black run-length | Code-word |
|---|---|---|---|
| 0 | 00110101 | 0 | 0000110111 |
| 1 | 000111 | 1 | 010 |
| 2 | 0111 | 2 | 11 |
| 3 | 1000 | 3 | 10 |
| 4 | 1011 | 4 | 011 |
| 5 | 1100 | 5 | 0011 |
| 6 | 1110 | 6 | 0010 |
| 7 | 1111 | 7 | 00011 |
| 8 | 10011 | 8 | 000101 |
| 9 | 10100 | 9 | 000100 |
| 10 | 00111 | 10 | 0000100 |
| 11 | 01000 | 11 | 0000101 |
| 12 | 001000 | 12 | 0000111 |
| 13 | 000011 | 13 | 00000100 |
| 14 | 110100 | 14 | 00000111 |
| 15 | 110101 | 15 | 00001 1000 |
| 16 | 101010 | 16 | 00001 0111 |
| 17 | 101011 | 17 | 00001 1000 |
| 18 | 0100111 | 18 | 00000 1000 |
| 19 | 0001100 | 19 | 00001 100111 |
| 20 | 0001000 | 20 | 00001 101000 |
| 21 | 0010111 | 21 | 00001 101100 |
| 22 | 0000011 | 22 | 00000 101111 |
| 23 | 0000100 | 23 | 00000 101000 |
| 24 | 0101000 | 24 | 00000 101111 |
| 25 | 0101011 | 25 | 00000 011000 |
| 26 | 0010011 | 26 | 00001 1001010 |
| 27 | 0100100 | 27 | 00001 1001011 |
| 28 | 0011000 | 28 | 00001 1001100 |
| 29 | 00000010 | 29 | 00001 1001101 |
| 30 | 00000011 | 30 | 00000 1010000 |
| 31 | 00011010 | 31 | 00000 1010001 |
| 32 | 00011011 | 32 | 00000 1010010 |
| 33 | 00010010 | 33 | 00000 1010011 |
| 34 | 00010011 | 34 | 00001 1010010 |
| 35 | 00010100 | 35 | 00001 1010011 |
| 36 | 00010101 | 36 | 00001 1010100 |
| 37 | 00010110 | 37 | 00001 1010101 |
| 38 | 00010111 | 38 | 00001 1010110 |
| 39 | 00101000 | 39 | 00001 1010111 |
| 40 | 00101001 | 40 | 00000 1011100 |
| 41 | 00101011 | 41 | 00000 1011101 |
| 42 | 00101011 | 42 | 00001 1011010 |
| 43 | 00101100 | 43 | 00001 1011011 |
| 44 | 00101101 | 44 | 00000 1010100 |
| 45 | 00000100 | 45 | 00000 1010101 |
| 46 | 00000101 | 46 | 00000 1010110 |
| 47 | 00001010 | 47 | 00000 1010111 |
| 48 | 00001011 | 48 | 00000 1100100 |
| 49 | 01010010 | 49 | 00000 1100101 |
| 50 | 01010011 | 50 | 00000 1010010 |
| 51 | 01010100 | 51 | 00000 1010011 |
| 52 | 01010101 | 52 | 00000 1100100 |
| 53 | 00100100 | 53 | 00000 1101011 |
| 54 | 00100101 | 54 | 00000 1101000 |
| 55 | 01011000 | 55 | 00000 1100111 |

| White run-length | Code-word | Black run-length | Code-word |
|---|---|---|---|
| 56 | 01011001 | 56 | 00000 1101000 |
| 57 | 01011010 | 57 | 00000 1011000 |
| 58 | 01011011 | 58 | 00000 1011001 |
| 59 | 01001010 | 59 | 00000 1010111 |
| 60 | 01001011 | 60 | 00000 1011100 |
| 61 | 00110010 | 61 | 00000 1011010 |
| 62 | 00110011 | 62 | 00000 1100110 |
| 63 | 00110100 | 63 | 00000 1100111 |

| White run-length | Code-word | Black run-length | Code-word |
|---|---|---|---|
| 64 | 11011 | 64 | 0000001111 |
| 128 | 10010 | 128 | 00001 1001000 |
| 192 | 010111 | 192 | 00001 1001001 |
| 256 | 0110111 | 256 | 00000 1011011 |
| 320 | 00110110 | 320 | 00000 1100011 |
| 384 | 00110111 | 384 | 00000 1100100 |
| 448 | 01100100 | 448 | 00000 1100101 |
| 512 | 01100101 | 512 | 00000 1101100 |
| 576 | 01101000 | 576 | 00000 1101101 |
| 640 | 01100111 | 640 | 00000 1001010 |
| 704 | 011001100 | 704 | 00000 1001011 |
| 768 | 011001101 | 768 | 00000 1001100 |
| 832 | 011010010 | 832 | 00000 1001101 |
| 896 | 011010011 | 896 | 00000 1110010 |
| 960 | 011010100 | 960 | 00000 1110011 |
| 1024 | 011010101 | 1024 | 00000 1110100 |
| 1088 | 011010110 | 1088 | 00000 1110101 |
| 1152 | 011010111 | 1152 | 00000 1110110 |
| 1216 | 011011000 | 1216 | 00000 1110111 |
| 1280 | 011011001 | 1280 | 00000 1010010 |
| 1344 | 011011010 | 1344 | 00000 1010011 |
| 1408 | 011011011 | 1408 | 00000 1010100 |
| 1472 | 010011000 | 1472 | 00000 1010101 |
| 1536 | 010011001 | 1536 | 00000 1011010 |
| 1600 | 010011010 | 1600 | 00000 1011011 |
| 1664 | 011000 | 1664 | 00000 1100100 |
| 1728 | 010011011 | 1728 | 00000 1100101 |
| 1792 | 00000001000 | 1792 | 00000001000 |
| 1856 | 00000001100 | 1856 | 00000001100 |
| 1920 | 00000001101 | 1920 | 00000001101 |
| 1984 | 00000010010 | 1984 | 00000010010 |
| 2048 | 00000010011 | 2048 | 00000010011 |
| 2112 | 00000010100 | 2112 | 00000010100 |
| 2176 | 00000010101 | 2176 | 00000010101 |
| 2240 | 00000010110 | 2240 | 00000010110 |
| 2304 | 00000010111 | 2304 | 00000010111 |
| 2368 | 00000011100 | 2368 | 00000011100 |
| 2432 | 00000011101 | 2432 | 00000011101 |
| 2496 | 00000011110 | 2496 | 00000011110 |
| 2560 | 00000011111 | 2560 | 00000011111 |
| EOL | 000000000001 | EOL | 000000000001 |

← Make-up codes

$$r_l = 64 \cdot m + t$$

Runlength of 12 white pixels

0010000

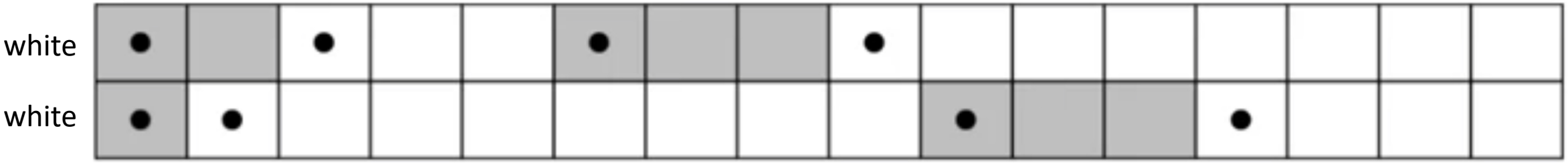
Runlength of 140 black pixels

140 = 2 x 64 + 12

000011001000000000111

# G3, G4 Facsimile Standards

## 2-D codes




1st row: 0, 2, 3, 3, 8 1, 3, 6, 9

2nd row: 0, 1, 8, 3, 4 1, 2, 10, 13

- 2-D scheme (modified READ – MR)
  - READ: Relative Element Address Designate
  - Exploit heavy correlation of image rows
  - Code transition points with reference to previous line, or distance from previous point is same line

# Lossless Compression

- Introduction
- Elements of Information Theory – Part I
- Elements of Information Theory – Part II
- Huffman Coding
- Run-Length Coding and Fax
- **Arithmetic Coding**
- Dictionary Techniques
- Predictive Coding

# Arithmetic Coding

- It is more efficient to generate codewords for groups or sequences of symbols rather than generating a separate codeword for each symbol in the sequence
- With Huffman encoding, in finding a codeword for a *particular* sequence of length m, we need codewords for all possible sequences of length m
- A way to assign codewords to *particular* sequences without having to generate codes for all sequences of that length, is offered by **arithmetic coding**
- In arithmetic coding a unique identifier (or **tag**) is generated for the sequence to be encoded, which corresponds to a binary fraction, which becomes the binary code for the sequence

# Generating a Tag

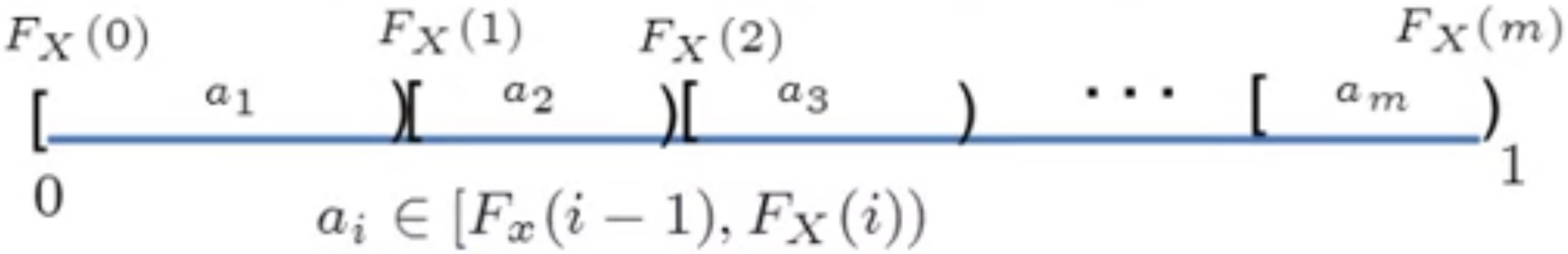
$A = \{a_1, a_2, \dots, a_m\}; P(a_i)$ Random variable: $X(a_i) = i$

Probability density function (pdf): $P(X = i) = P(a_i)$

Cumulative distribution function (cdf): $F_X(i) = \sum_{k=1}^{i} P(X = k)$

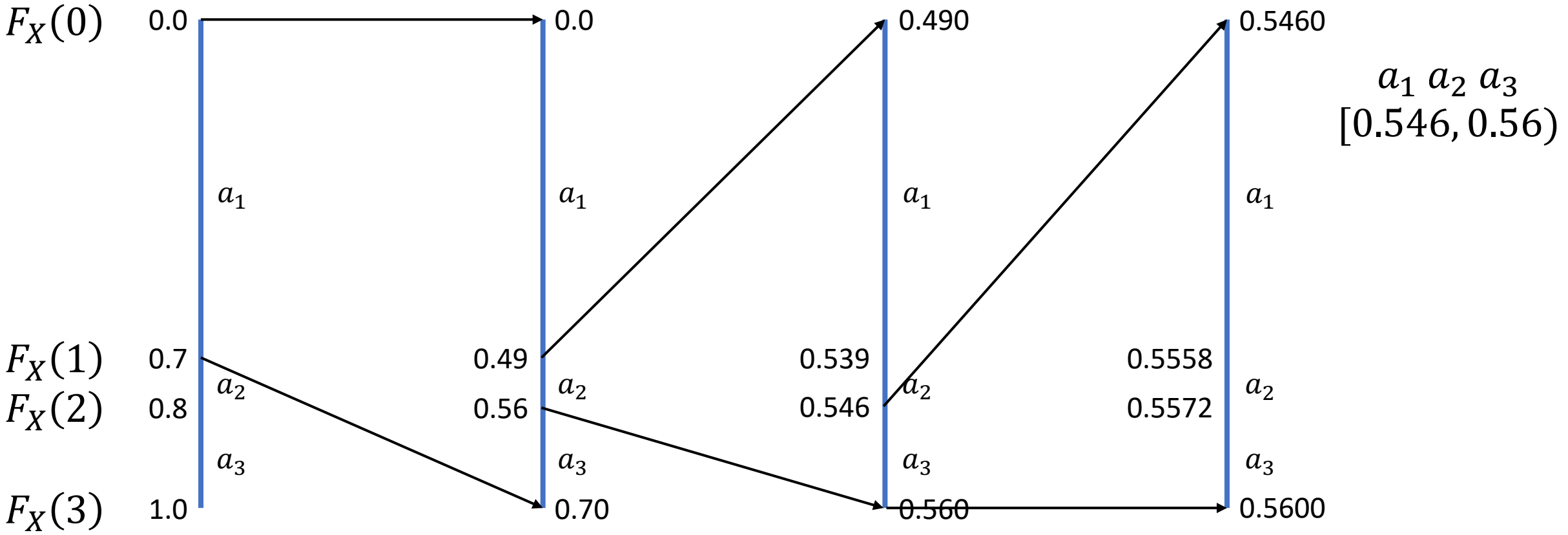
**IDEA**: Reduce the size of the interval in which the tag resides as more and more elements of the sequence are received

Partition the unit into sub-interval of the form



# Arithmetic Coding Example

$A = \{a_1, a_2, a_3\}; P(a_1) = 0.7, p(a_2) = 0.1, P(a_3) = 0.2; F_X(1) = 0.7, F_X(2) = 0.8, F_X(3) = 1$

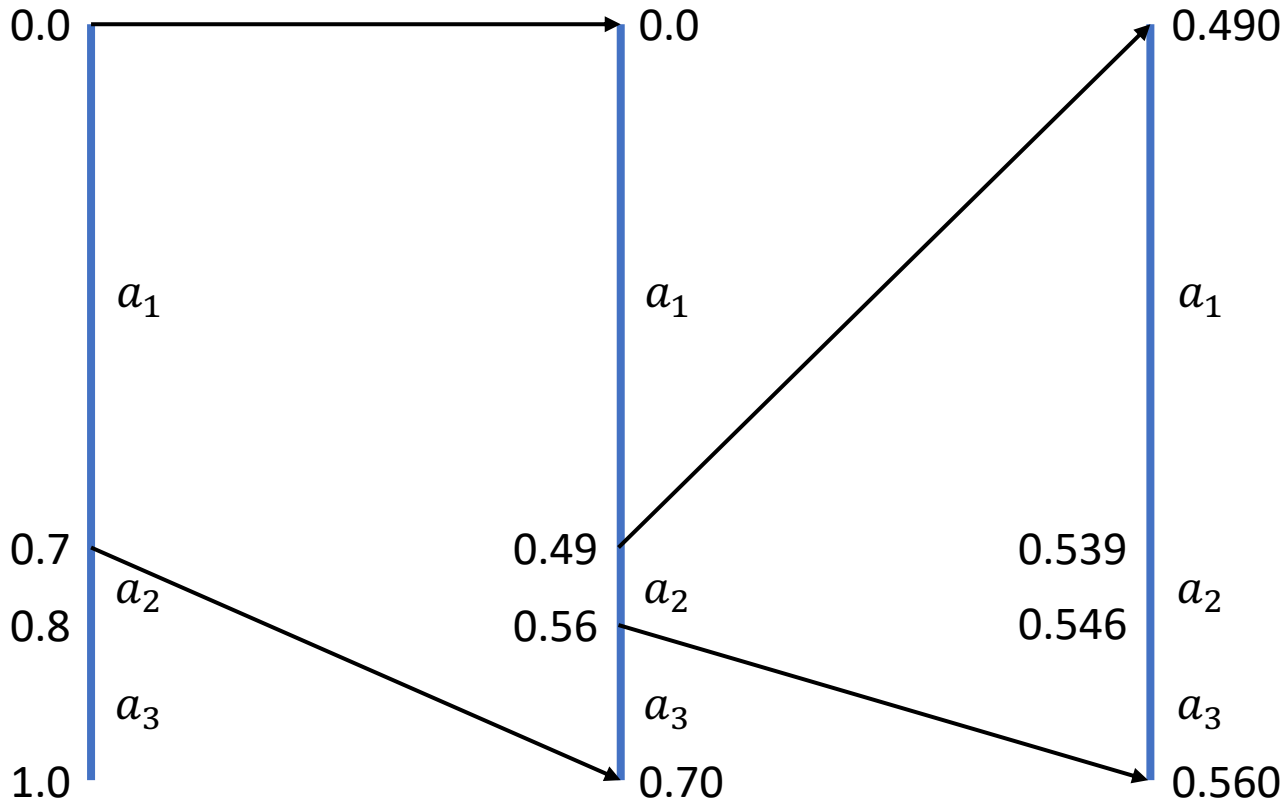


# Deciphering the Tag

$A = \{a_1, a_2, a_3\}; P(a_1) = 0.7, p(a_2) = 0.1, P(a_3) = 0.2; F_X(1) = 0.7, F_X(2) = 0.8, F_X(3) = 1$

Tag = 0.55
3 symbols

$a_1 \ a_2 \ a_3$




Mimic the encoder for decoding

# AC vs Huffman

$$H(S) \leq R_{\text{Huffman}} < H(S) + \frac{1}{N}$$

$$H(S) \leq R_{\text{AC}} < H(S) + \frac{2}{N}$$

Scaling and incremental coding

# Taking Context into Account

Assume a given document has: $P(B) = 0.2, P(W) = 0.8$

$$H = -0.2 \log 0.2 - 0.8 \log 0.8 = 0.722 \text{ bpp}$$

Divide document into 2 regions:

Region 1 includes 80% of pixels

$P(W) = 0.95, P(B) = 0.05$

$H_1 = 0.286 \text{ bpp}$

Region 2 includes 20% of pixels

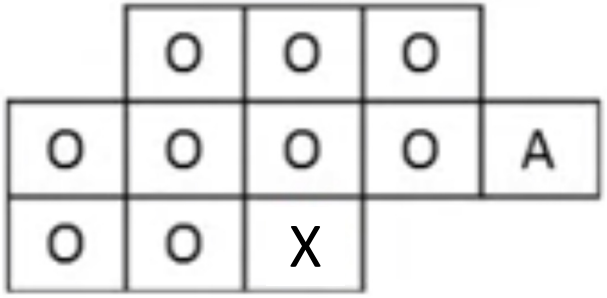
$P(B) = 0.7, P(W) = 0.3$

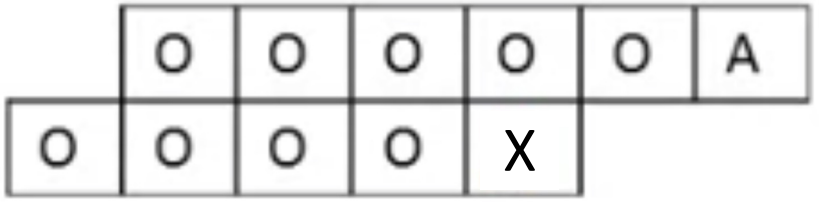
$H_2 = 0.881 \text{ bpp}$

Total Entropy: $H = 0.8H_1 + 0.2H_2 = 0.405 \text{ bpp}$

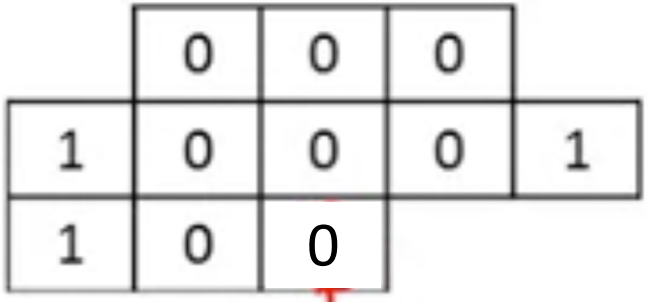
# JBIG

- Joint Bi-Level Image Processing Group (JBIG)
- Standard for the progressive transmission of bi-level images




10-pixel neighborhoods



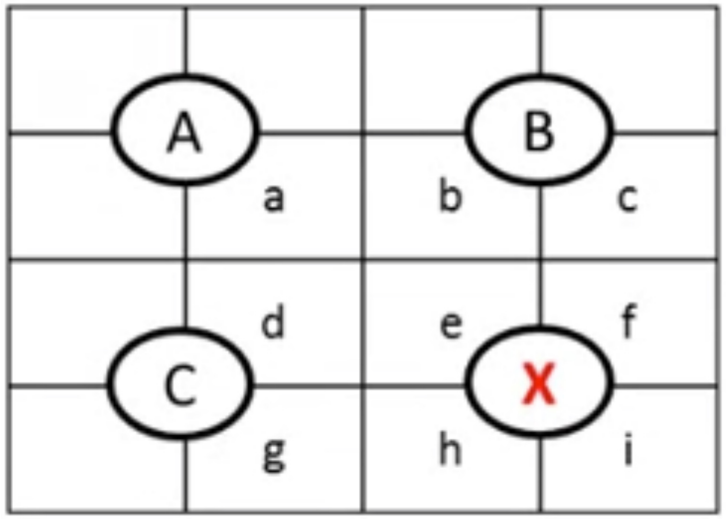

# JBIG-Progressive Transmission

| | |
|---|---|
| 0 | 0 |
| 0 | 1 |

1/4 → 0

| | |
|---|---|
| 0 | 0 |
| 1 | 1 |

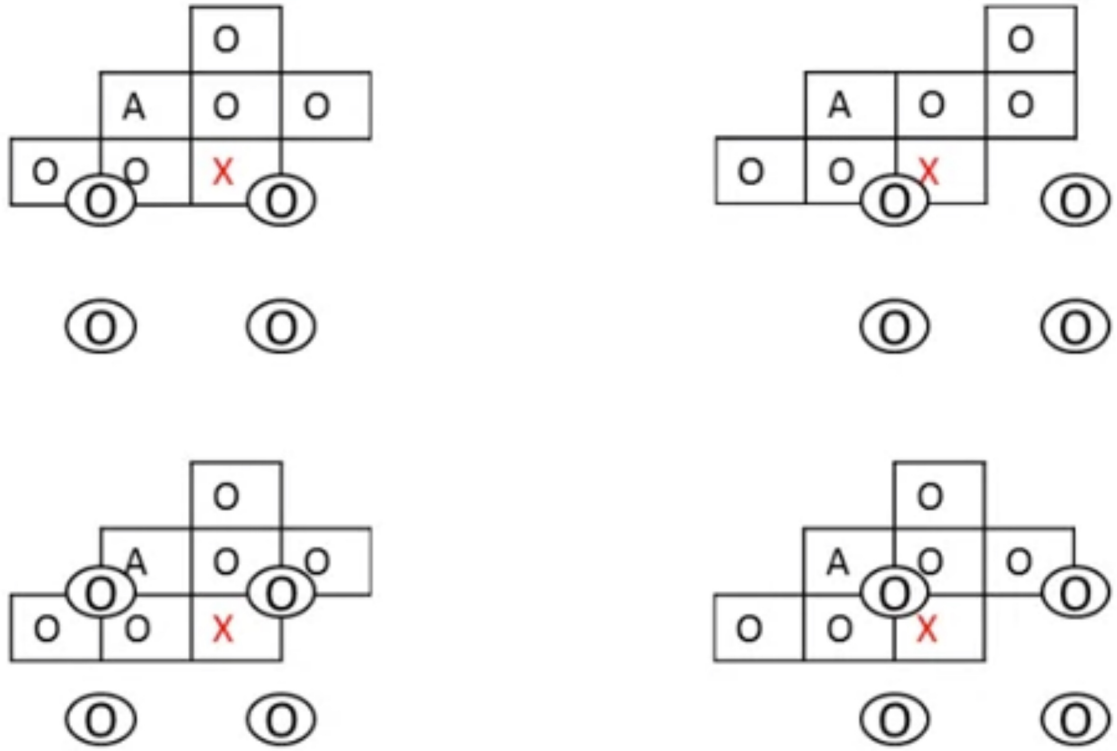
1/2 → 0? 1?




Pixels used for downsampling

$$4e + 2(b + d + f + h) + (a + c + g + i) - 3(B + C) - A \begin{cases} > 4.5 \Rightarrow X = 1 \\ < 4.5 \Rightarrow X = 0 \end{cases}$$

# JBIG-Lossless Compression Revisited




Context templates used in the coding of higher-resolution layers

# Lossless Compression

- Introduction
- Elements of Information Theory – Part I
- Elements of Information Theory – Part II
- Huffman Coding
- Run-Length Coding and Fax
- Arithmetic Coding
- **Dictionary Techniques**
- Predictive Coding

# Dictionary Techniques

- In many applications, the output of the source consists of recurring patterns
- A very reasonable approach to encode such sources is to keep a list, or *dictionary*, of frequently occurring patterns.
- The input is split into two classes, frequently occurring and infrequently occurring patterns
- There are static and adaptive dictionary techniques
- Most adaptive techniques have their roots in two papers by Ziv and Lempel in 1977 (LZ77) and 1978 (LZ78)

# Static Dictionary

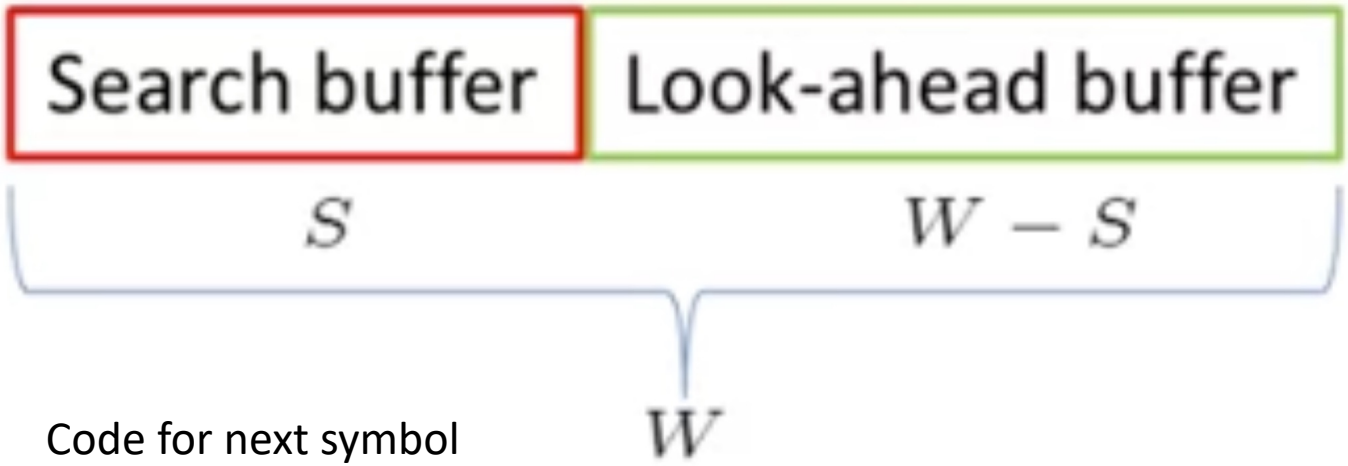
| Code | Entry |
|---|---|
| 000 | a |
| 001 | b |
| 010 | c |
| 011 | d |
| 100 | r |
| 101 | ab |
| 110 | ac |
| 111 | ad |

$$A = \{a, b, c, d, r\}$$
$$a\ b\ r\ a\ c\ a\ d\ a\ b\ r\ a$$

11 x 3 = 33 bits

7 x 3 = 21 bits

# LZ77 Approach




Code for next symbol

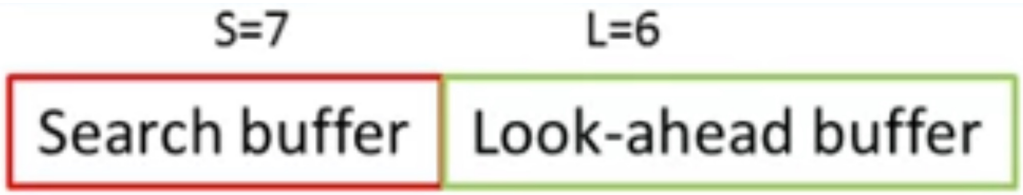
Encode: $\langle o, l, c \rangle$

offset  Length of match

Bits needed (using fixed length codes)

$$\lceil \log_2 S \rceil + \lceil \log_2 W \rceil + \lceil \log_2 |A| \rceil$$

# LZ77 Example




c a b r a c a d a b r a r r a r r a d

$\langle o,\ l,\ c \rangle$

Example: Encoding

c a b r a c a d a b r a r r a r r a d

c a b r a c a d a b r a r r a r r a d

c a b r a c a d a b r a r r a r r a d

Example: Decoding

c a b r a c a

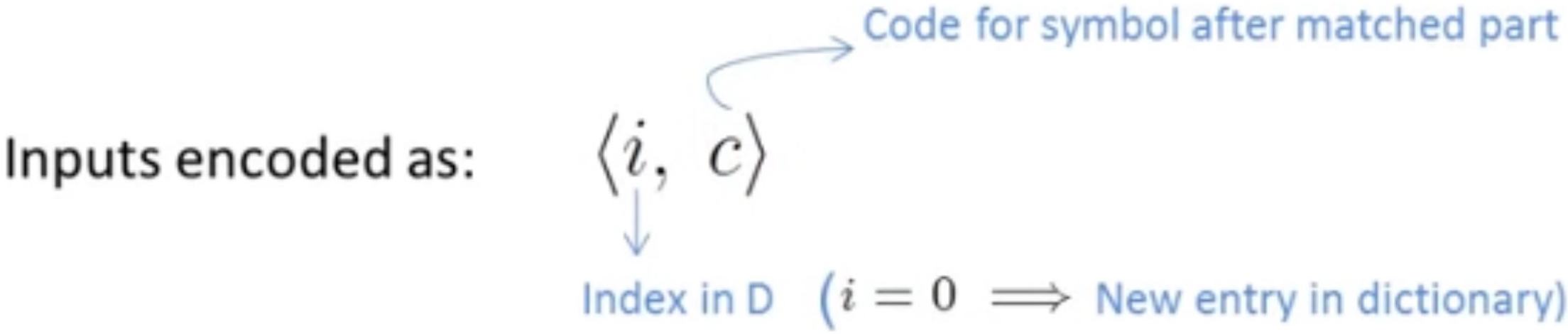
c a b r a c a d

c a b r a c a d a b r a r

c a b r a c a d a b r a r r a r r a d

# LZ78

- Drops reliance on search buffer and keeps an EXPLICIT dictionary
- This dictionary has to be built as both the encoder and decoder, and therefore they have to be identical



# LZ78 Example

b a w w a / b a w w a / b a w w a / b a w w a / b o o / b o

| Encoder output | Index | Entry |
|---|---|---|
| <0,c(b)> | 1 | b |
| <0,c(a)> | 2 | a |
| <0,c(w)> | 3 | w |
| <3,c(a)> | 4 | wa |
| <0,c(/)> | 5 | / |
| <1,c(a)> | 6 | ba |
| <3,c(w)> | 7 | ww |
| <2,c(/)> | 8 | a/ |
| <6,c(w)> | 9 | baw |
| <4,c(/)> | 10 | wa/ |
| <9,c(w)> | 11 | baww |
| <8,c(b)> | 12 | a/b |
| <0,c(o)> | 13 | o |
| <13,c(/)> | 14 | o/ |
| <1,c(o)> | 15 | bo |

# Dictionary Techniques

- Variations of LZ77
  - Efficient encoding of the triplets
  - Variable size of S and L buffers
  - Eliminate 3rd element in triplet by adding a flag when a single symbol is encountered
- Variation of LZ78
  - LZW: encoder only sends index
- Applications of LZx
  - Unix *compress*
  - GIF (Graphics Interchange Format)
  - Compression over modems – V.42 bis

# Lossless Compression

- Introduction
- Elements of Information Theory – Part I
- Elements of Information Theory – Part II
- Huffman Coding
- Run-Length Coding and Fax
- Arithmetic Coding
- Dictionary Techniques
- **Predictive Coding**

# Predictive Coding


**Reconstruction = Prediction + Prediction Error**

- Context Adaptive Lossless Image Compression (CALIC)
- Joint Photographic Experts Group (JPEG)-LS

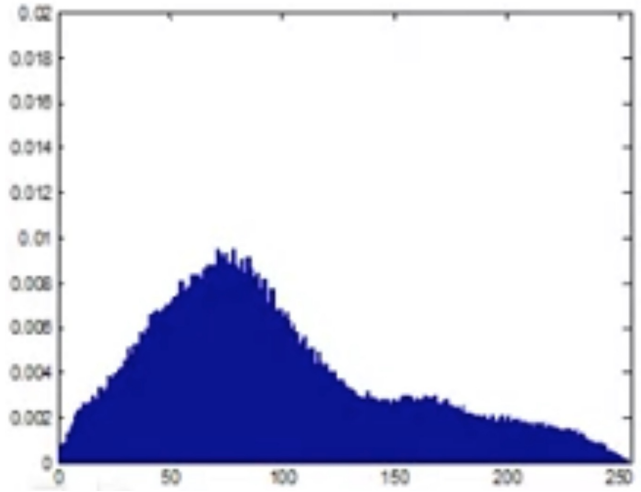
# Prediction Example

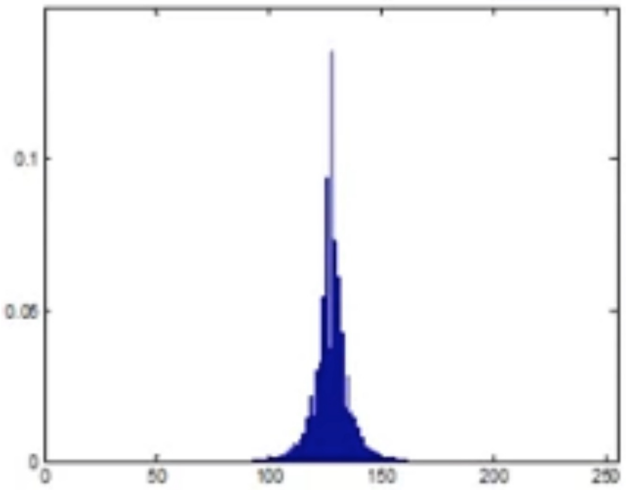
predicted value of $x(n)$:

$$x_p(n) = x(n-1)$$

prediction error:

$$e(n) = x(n) - x_p(n)$$
$$= x(n) - x(n-1)$$



# JPEG-LS Predictors

Predictors:

$$
\begin{aligned}
&0 \text{ No predictor} \\
&1\ \hat{f}(i,j) = f(i-1,j) \\
&2\ \hat{f}(i,j) = f(i,j-1) \\
&3\ \hat{f}(i,j) = f(i-1,j-1) \\
&4\ \hat{f}(i,j) = f(i,j-1) + f(i-1,j) - f(i-1,j-1) \\
&5\ \hat{f}(i,j) = f(i,j-1) + (f(i-1,j) - f(i-1,j-1))/2 \\
&6\ \hat{f}(i,j) = f(i-1,j) + (f(i,j-1) - f(i-1,j-1))/2 \\
&7\ \hat{f}(i,j) = (f(i,j-1) + f(i-1,j))/2
\end{aligned}
$$

Correction encoded by adaptive arithmetic coding

# JPEG-LS Example

| Image | JPEG 0 | JPEG 1 | JPEG 2 | JPEG 3 | JPEG 4 | JPEG 5 | JPEG 6 | JPEG 7 |
|---|---|---|---|---|---|---|---|---|
| Sena | 53,431 | 37,220 | 31,559 | 38,261 | 31,055 | **29,742** | 33,063 | 32,179 |
| Sensin | 58,306 | 41,298 | 37,126 | 43,445 | **32,429** | 33,463 | 35,965 | 36,428 |
| Earth | 38,248 | 32,295 | **32,137** | 34,089 | 33,570 | 33,057 | 33,072 | 32,672 |
| Omaha | 56,061 | **48,818** | 51,283 | 53,909 | 53,771 | 53,520 | 52,542 | 52,189 |

Compressed file size in bytes of the residual images obtained using the various JPEG prediction modes

| Image | Best JPEG-LS | GIF |
|---|---|---|
| Sena | 31,055 | 51,085 |
| Sensin | 32,429 | 60,649 |
| Earth | 32,137 | 34,276 |
| Omaha | 48,818 | 61,341 |

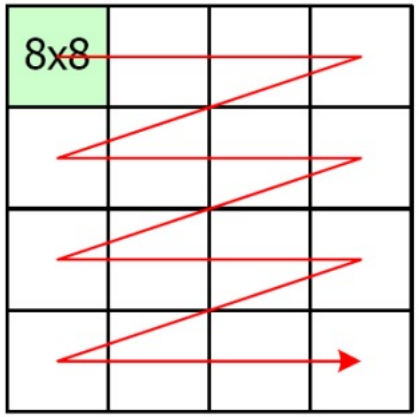
Comparison of the file sizes obtained using JPEG lossless compression and GIF

# Lossless Compression

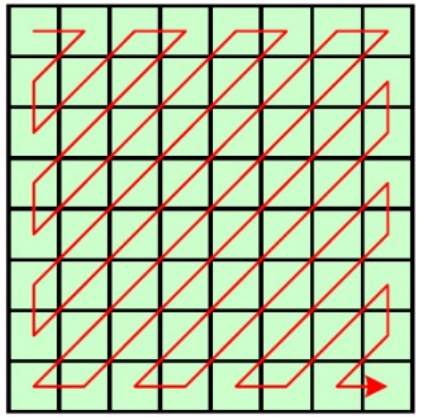
- Introduction
- Elements of Information Theory – Part I
- Elements of Information Theory – Part II
- Huffman Coding
- Run-Length Coding and Fax
- Arithmetic Coding
- Dictionary Techniques
- Predictive Coding

# Homework #3 – Entropy Coding

- Run length encoding and run length decoding
  - 8x8 block-based DCT coefficients of “lena.png”.
  - Quantize the coefficients with 16-bit for DC and 8-bit for AC.
  - Use raster scan to visit all 8x8 blocks in these images.
  - Do the run length encoding by using zigzag scan to visit all pixels in one block.
  - Do the run length decoding and IDCT to recover the image.
- Deadline: 2023/04/10 11:59 PM
- Compressed as a single ZIP file.
  Upload to E3 with filename:
  VC_HW3_[student_id].zip
- Required files :
  - Report PDF
  - Source code (C/C++/Python/MATLAB)



Block level scan


Pixel level scan

Ref: https://q-viper.github.io/2021/05/24/coding-run-length-encoding-in-python/