Image Compression

1. What and why image compression
2. Basic concepts
3. Encoding/decoding, entropy

What is Data and Image Compression?

- Data compression is the art and science of representing information in a compact form.
- Data is a sequence of symbols taken from a discrete alphabet.

Why do we need Image Compression?

Still Image

- One page of A4 format at 600 dpi is > 100 MB.
- One color image in digital camera generates 10-30 MB.
- Scanned 3”$\times$7” photograph at 300 dpi is 30 MB.

Digital Cinema

- 4K$\times$2K$\times$3 $\times$12 bits/pel = 48 MB/frame or 1 GB/sec or 70 GB/min.

Why do we need Image Compression?

1) Storage
2) Transmission
3) Data access

1990-2000

Disc capacities	: 100MB -> 20 GB (200 times!)
but seek time	: 15 milliseconds → 10 milliseconds
and transfer rate	: 1MB/sec ->2 MB/sec.

Compression improves overall response time in some applications.

Source of images

- Image scanner
- Digital camera
- Video camera,
- Ultra-sound (US), Computer Tomography (CT),
  Magnetic resonance image (MRI), digital X-ray (XR),
  Infrared.
- *etc*.

Image types

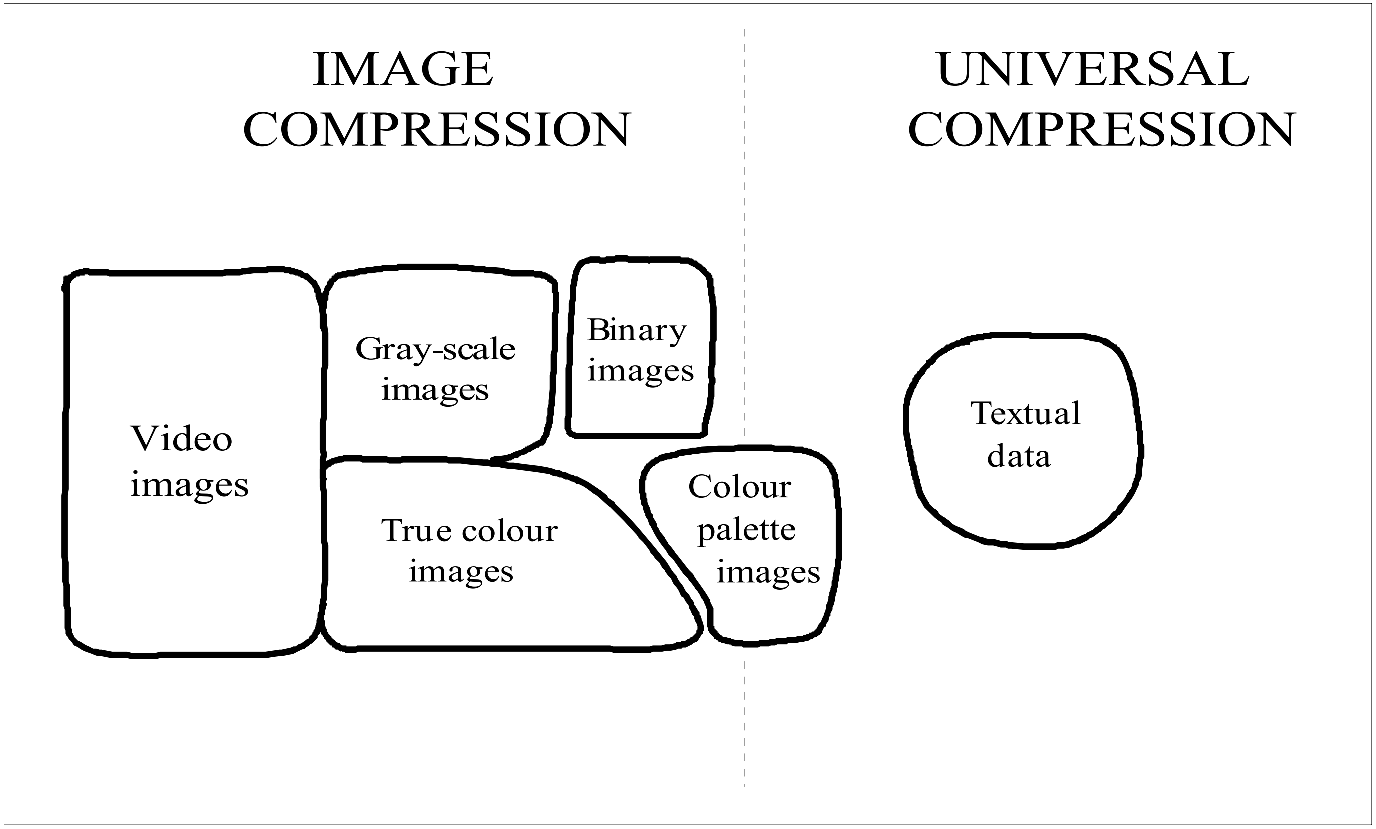


IMAGE COMPRESSION
UNIVERSAL COMPRESSION
Video images
Gray-scale images
Binary images
True colour images
Colour palette images
Textual data

Why we can compress image?

- Statistical redundancy:
  1) Spatial correlation
     a) Local - Pixels at neighboring locations have similar intensities.
     b) Global - Reoccurring patterns.
  2) Spectral correlation – between color planes.
  3) Temporal correlation – between consecutive frames.
- Tolerance to fidelity:
  1) Perceptual redundancy.
  2) Limitation of rendering hardware.

Lossy vs. Lossless compression

Lossless compression: reversible, information preserving
text compression algorithms,
binary images, palette images

Lossy compression: irreversible
grayscale, color, video

Near-lossless compression:
medical imaging, remote sensing.

Rate measures

Bitrate: $$\frac{\text{size of the compressed file}}{\text{pixels in the image}} = \frac{C}{N}$$ bits/pel

Compression ratio: $$\frac{\text{size of the original file}}{\text{size of the compressed file}} = \frac{N \cdot k}{C}$$

Distortion measures

Mean average error (MAE): $$\mathrm{MAE} = \frac{1}{N}\sum_{i=1}^{N}\left|y_i - x_i\right|$$

Mean square error (MSE): $$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - x_i\right)^2$$

Signal-to-noise ratio (SNR): $$\mathrm{SNR} = 10\cdot\log_{10}\left[\sigma^2/\mathrm{MSE}\right]$$

(decibels)

Pulse-signal-to-noise ratio (PSNR): $$\mathrm{PSNR} = 10\cdot\log_{10}\left[A^2/\mathrm{MSE}\right]$$

(decibels)

A is amplitude of the signal: A = 2^8-1=255 for 8-bits signal.

Other issues

- Coder and decoder computation complexity
- Memory requirements
- Fixed rate or variable rate
- Error resilience
- Symmetric or asymmetric
- Decompress at multiple resolutions
- Decompress at various bit rates
- Standard or proprietary

Entropy

Set of symbols (alphabet) $S=\{s_1, s_2, \ldots, s_N\}$,
N is number of symbols in the alphabet.

Probability distribution of the symbols: $P=\{p_1, p_2, \ldots, p_N\}$

According to Shannon, the entropy H of an information source S is defined as follows:

$$H = -\sum_{i=1}^{N} p_i \cdot \log_2(p_i)$$

Entropy

The amount of information in symbol s_i,

in other words, the number of bits to code or code length

for the symbol s_i:

$$H(s_i) = -\log_2(p_i)$$

The average number of bits for the source S:

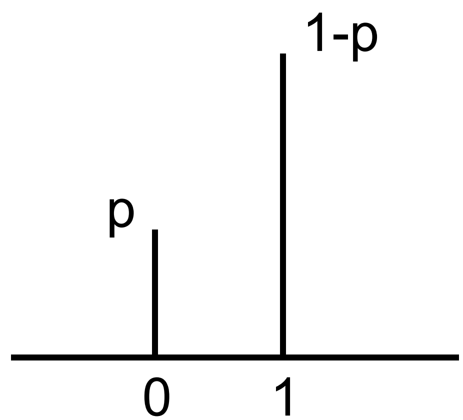
$$H = -\sum_{i=1}^{N} p_i \cdot \log_2(p_i)$$

Entropy for binary source: N=2

S={0,1}

p_0=p

p_1=1-p



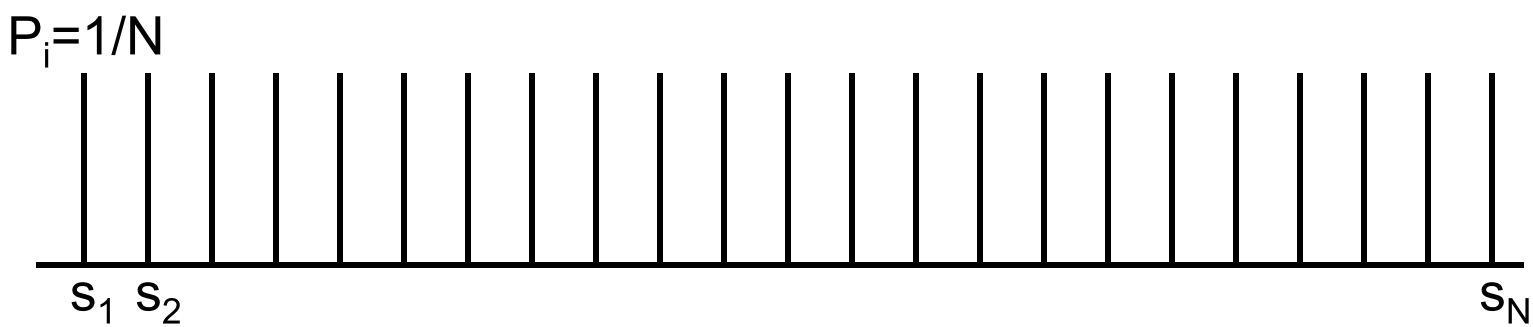
$$H = -(p \cdot \log_2 p + (1-p) \cdot \log_2 (1-p))$$

H=1 bit   for p_0=p_1=0.5

Entropy for uniform distribution: $p_i=1/N$

Uniform distribution of probabilities: $p_i=1/N$:

$$H=-\sum_{i=1}^{N}(1/N)\cdot\log_2(1/N)=\log_2(N)$$



Examples:

N= 2: $p_i=0.5$; $H=\log_2(2) = 1$ bit

N=256: $p_i=1/256$; $H=\log_2(256)= 8$ bits

How to get the probability distribution?

1) Static modeling:
   a) The same code table is applied to all input data.
   b) One-pass method (encoding)
   c) No side information
2) Semi-adaptive modeling:
   a) Two-pass method: (1) analysis and (2) encoding.
   b) Side information needed (model, code table)
3) Adaptive (dynamic) modeling:
   a) One-pass method: analysis and encoding
   b) Updating the model during encoding/decoding
   c) No side information

Static vs. Dynamic: Example

S = {a,b,c}; Data: a,a,b,a,a,c,a,a,b,a.

1) Static model: p_i=1/10

$$H = -\log_2(1/10)=\mathbf{1.58}\text{ bits}$$

2) Semi-adaptive method: p_1=7/10; p_2=2/10; p_3=1/10;

$$H = -(0.7*\log_2 0.7 + 0.2*\log_2 0.2 + 0.1*\log_2 0.1)=\mathbf{1.16}\text{ bits}$$

3) Adaptive method: Example

S = {a,b,c}; Data: a,a,b,a,a,c,a,a,b,a.

Symbol	1	2	3	4	5	6	7	8	9	10
a	**1**	**2**	3	**3**	**4**	5	**5**	**6**	7	**7**
b	1	1	**1**	2	2	2	2	2	**2**	3
c	1	1	1	1	1	**1**	2	2	2	2
p_i	0.33	0.5	0.2	0.5	0.57	0.13	0.56	0.60	0.18	0.58
H	1.58	1.0	2.32	1.0	0.81	3.0	0.85	0.74	2.46	0.78

H=(1/10)(1.58+1.0+2.32+1.0+0.81+3.0+0.85+0.74+2.46+0.78)
=**1.45** bits/char

1.16 < 1.45 < 1.58
S.-Ad. Ad. Static

Coding methods

- Shannon-Fano Coding
- Huffman Coding
- Predictive coding
- Block coding

- Arithmetic code
- Golomb-Rice codes

Shannon-Fano Code: A top-down approach

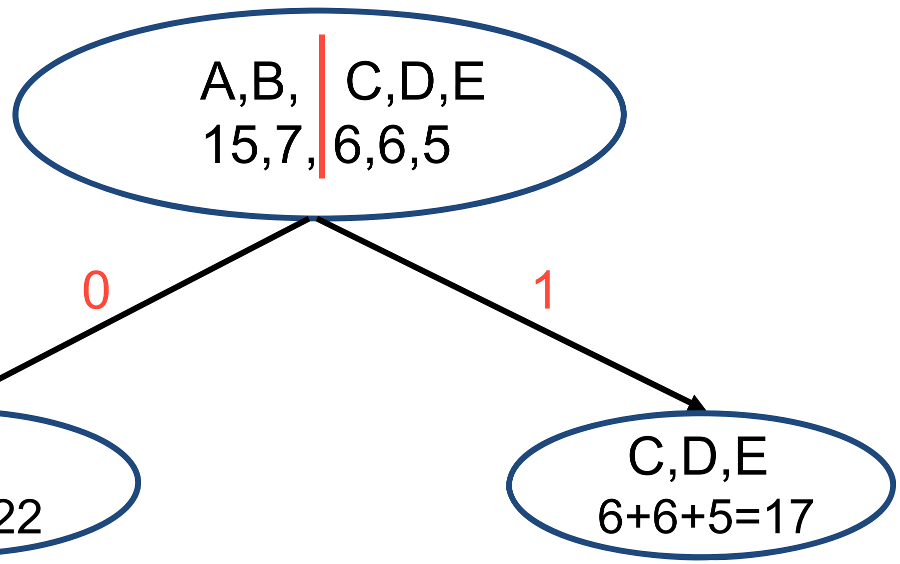
1) Sort symbols according their probabilities:

$$p_1 \le p_2 \le \ldots \le p_N$$

2) Recursively divide into parts, each with approx. the same number of counts (probability)

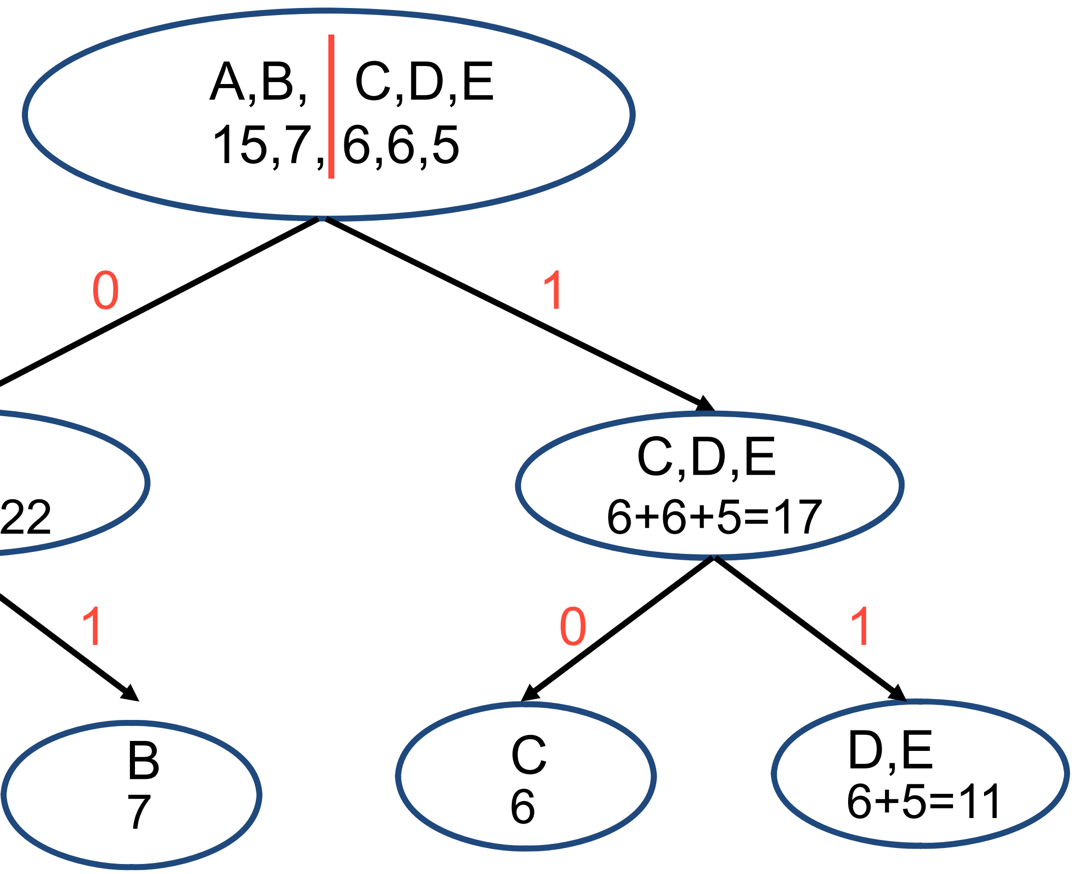
Shannon-Fano Code: Example (1 step)

s_i	p_i
A -	15/39
B -	7/39
C -	6/39
D -	6/39
E -	5/39



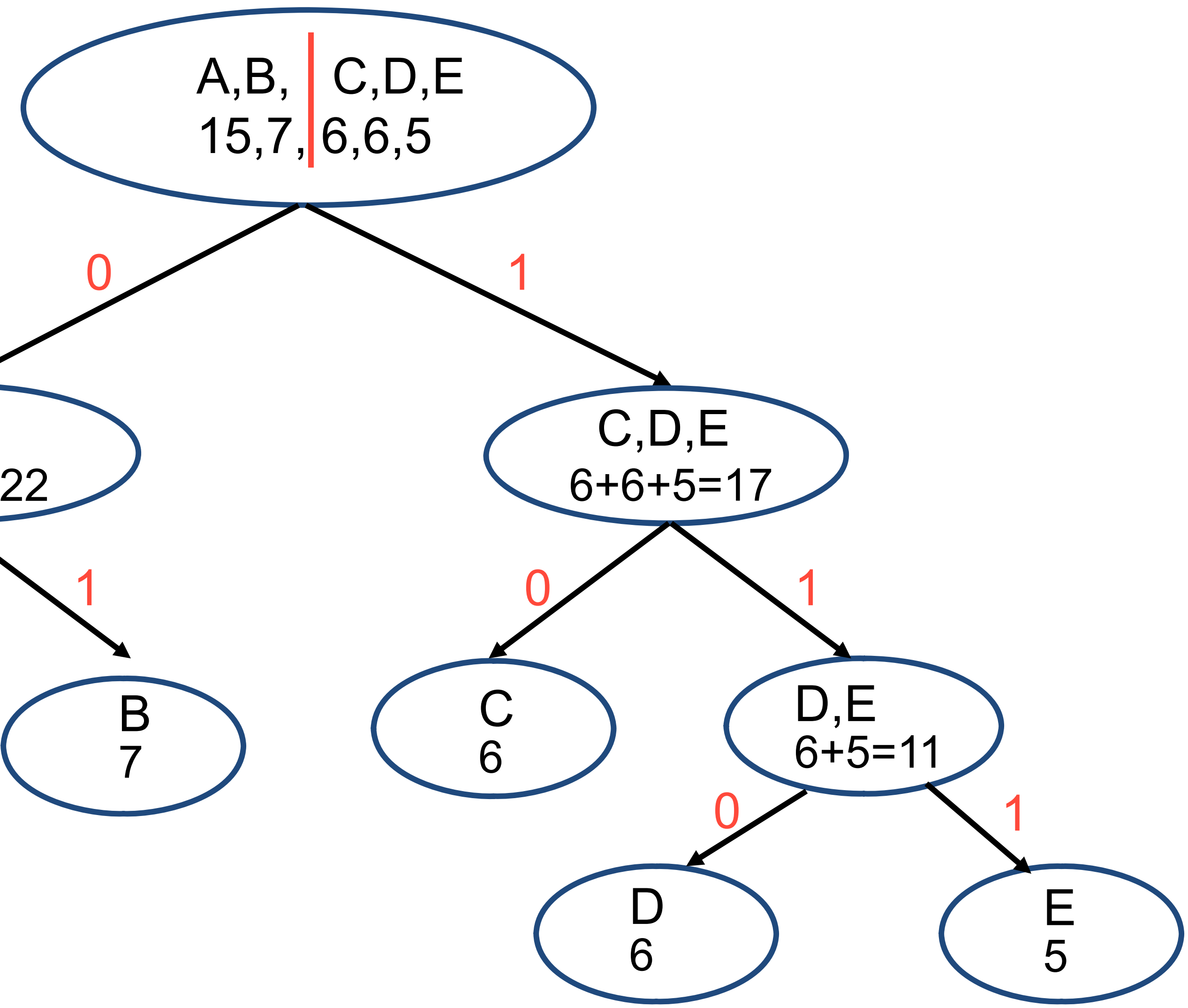
Shannon-Fano Code: Example (2 step)

s_i	p_i
A -	15/39
B -	7/39
C -	6/39
D -	6/39
E -	5/39



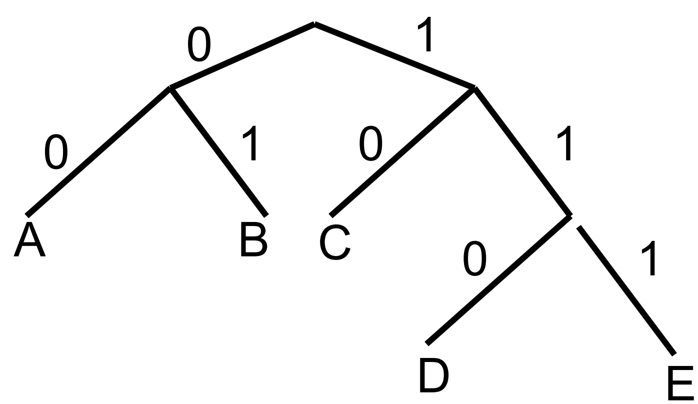
Shannon-Fano Code: Example (3 step)

s_i	p_i
A -	15/39
B -	7/39
C -	6/39
D -	6/39
E -	5/39



Shannon-Fano Code: Example (Result)

Symbol	p_i	$-\log_2(p_i)$	Code	Subtotal
A	15/39	1.38	00	2*15
B	7/39	2.48	01	2*7
C	6/39	2.70	10	2*6
D	6/39	2.70	110	3*6
E	5/39	2.96	111	3*5
Total:				89 bits



Binary tree

H=89/39=2.28 bits

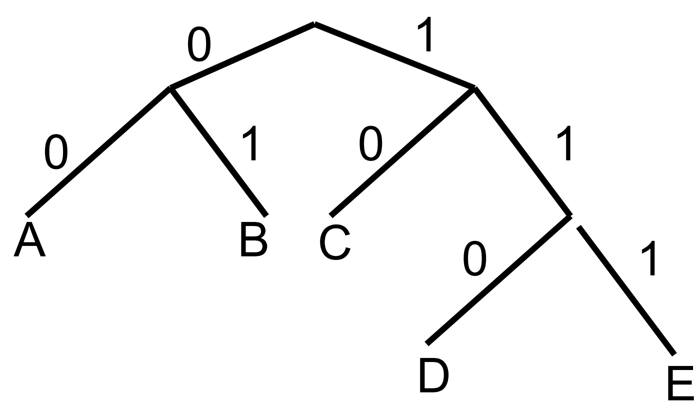
Shannon-Fano Code: Encoding

A - 00
B - 01
C - 10
D - 110
E - 111

Message: B A B A C A C A D E

Codes: 01 00 01 00 10 00 10 00 110 111

Bitstream: 0100010010001000110111



Binary tree

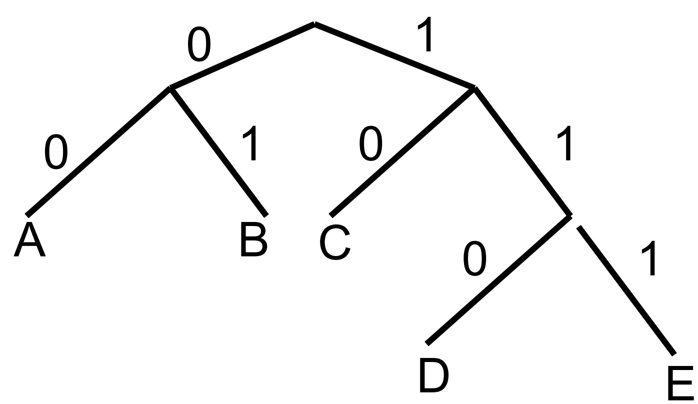
Shannon-Fano Code: Decoding

A - 00
B - 01
C - 10
D - 110
E - 111

Bitstream: 01000100100010000110111 (23 bits)

Codes: 01 00 01 00 10 00 10 00 110 111

Messaage: B A B A C A C A D E



Binary tree

Huffman Code: A bottom-up approach

INIT:
Put all nodes in an OPEN list, keep it sorted all times according their probabilities;.

REPEAT

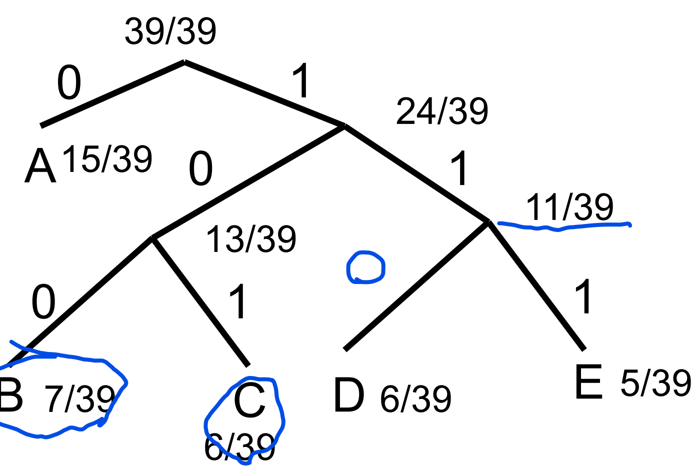
a) From OPEN pick *two* nodes having the *lowest* probabilities, create a parent node of them.

b) Assign the sum of the children's probabilities to the parent node and inset it into OPEN

c) Assign code 0 and 1 to the two branches of the tree, and delete the children from OPEN.

Huffman Code: Example

Symbol	p_i	$-\log_2(p_i)$	Code	Subtotal
A	15/39	1.38	0	2*15
B	7/39	2.48	100	3*7
C	6/39	2.70	101	3*6
D	6/39	2.70	110	3*6
E	5/39	2.96	111	3*5
Total:				87 bits



Binary tree

H=87/39=2.23 bits

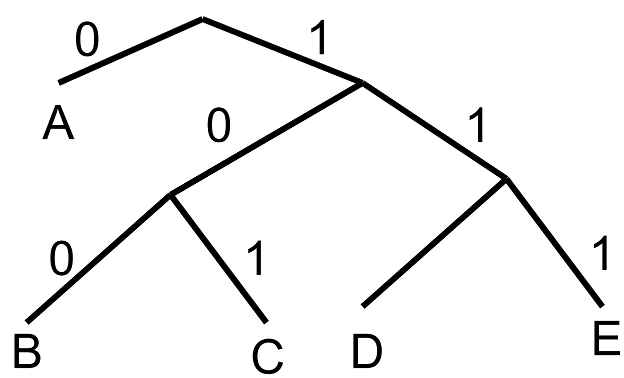
Huffman Code: Decoding

A - 0
B - 100
C - 101
D - 110
E - 111

Bitstream: 1000100010101010110111 (22 bits)

Codes: 100 0 100 0 101 0 101 0 110 111

Message: B A B A C A C A D E



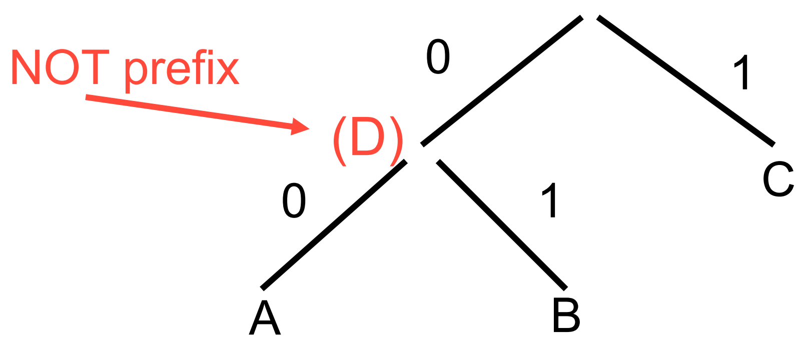
Binary tree

Properties of Huffman code

- Optimum code for a given data set requires two passes.
- Code construction complexity O(N logN).
- Fast lookup table based implementation.
- Requires at least one bit per symbol.
- Average codeword length is within one bit of zero-order entropy (Tighter bounds are known): $H \leq R < H+1$ bit
- Susceptible to bit errors.

Unique prefix property

No code is a prefix to any other code, all symbols are the *leaf* nodes



Shannon-Fano and Huffman codes are prefix codes

Legend: Shannon (1948) and Fano (1949);

Huffman (1952) was student of Fano at MIT.

Fano: ”Construct minimum-redundancy code → final exam is passed!”

Predictive coding

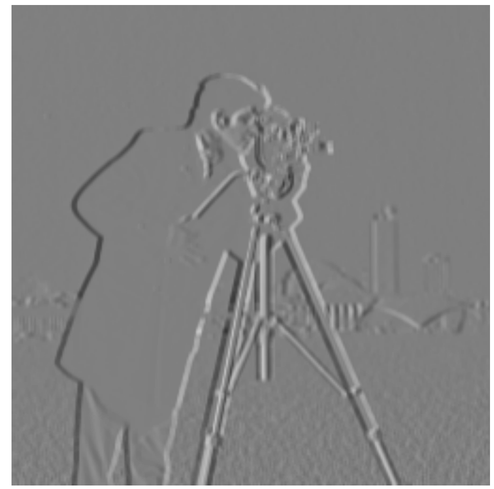
1) Calculate prediction value: y_i=f(neibourhood of x_i).

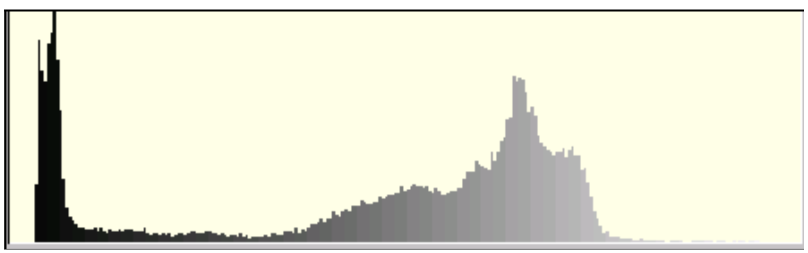
2) Calculating the prediction error: $e_i = y_i - x_i$.

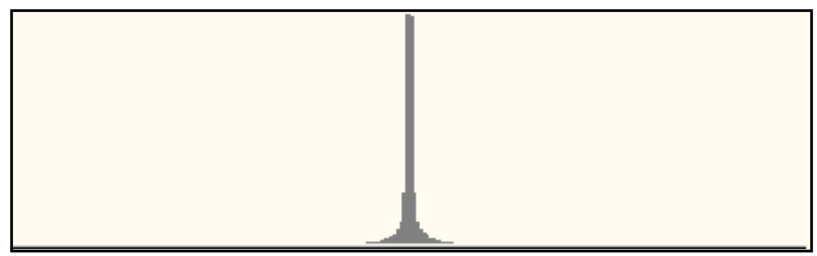
3) Encode the prediction error e_i.

Predictive model for grayscale images

$y = x_i - x_{i-1}$



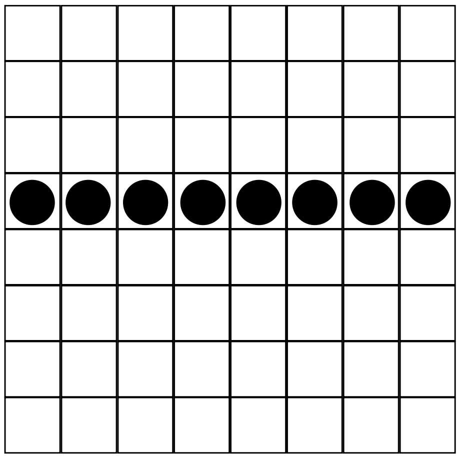




Histogram of the original image and Residual image

Entropy: H_o= 7.8 bits/pel (?) H_r=5.1 bits/pel (?)

Coding without prediction



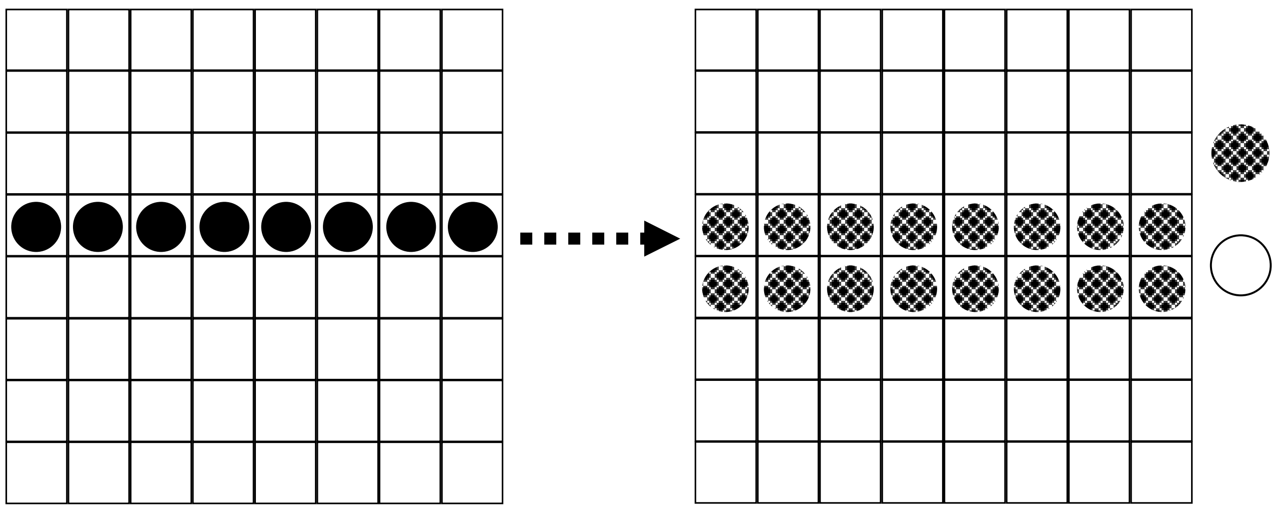
● f_0=8; p_0=p=8/64 =0.125;

○ f_1=56; p_1 =(1-p)=56/64=0.875

Entropy:

$$H = -((8/64)*\log_2(8/64)+(56/64)*\log_2(56/64))=\mathbf{0.544} \text{ bits/pel}$$

Prediction for binary images by pixel above



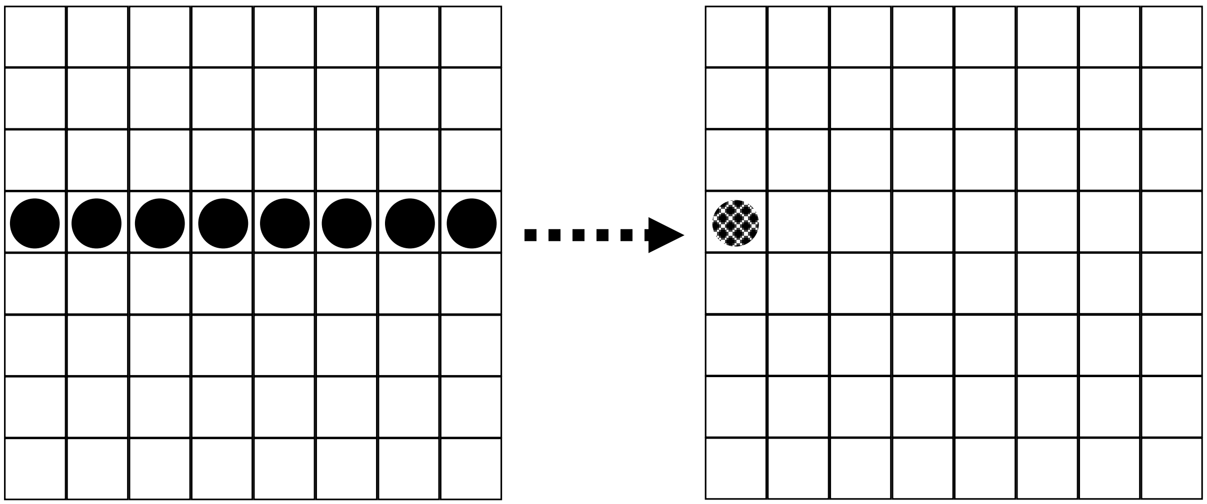
f	p
16	16/64
48	48/64

Entropy:

$$H = -((16/64)*\log_2(16/64)+(48/64)*\log_2(48/64)) = \mathbf{0.811}\ \text{bits/pel}$$

Wrong predictor!

Prediction for binary images pixel to the left



	f	p
●	1	16/64
○	63	63/64

Entropy:

$$H = -((1/64)*\log_2(1/64) + (63/64)*\log_2(63/64)) = \mathbf{0.116} \text{ bits/pel}$$

Good predictor!

Comparison of predictors:

- Without prediction: H= 0.544 bits/pel
- Prediction by pixel above: H = 0.811 bits /pel (bad!)
- Prediction by pixel to the left: H=**0.116** bits/pel (good!)