CSL7320: Digital Image Analysis

Image Compression

Image Compression

- Much of the information is graphical or pictorial in nature, the storage and communication requirements are immense.
- Image compression addresses the problem of reducing the amount of data requirements to represent a digital image.
- Image compression is becoming an enabling technology: HDTV.
- Also it plays an important role in transmission, video conferencing, remote sensing, satellite TV, document and medical imaging.

Slide credit: Ashish Ghosh

Why do we need compression?

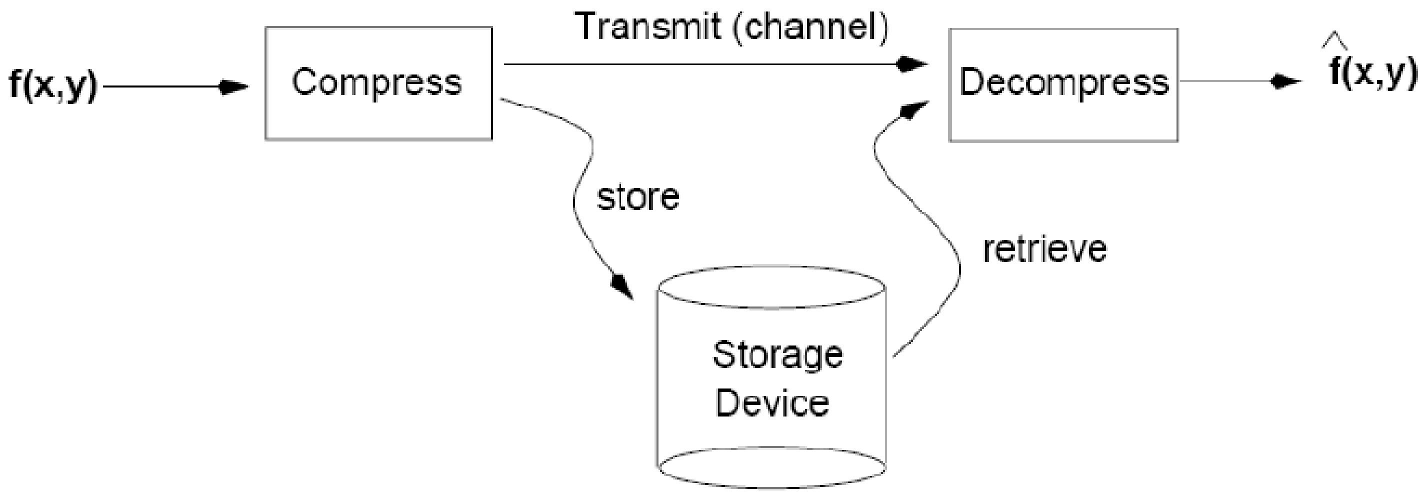
- For STORAGE and TRANSMISSION
  - DVD
  - Remote sensing
  - Video conferencing
  - Control of remotely piloted vehicle
- The bit rate of uncompressed digital cinema

  data exceeds 1 Gbps

Slide credit: Ashish Ghosh

Goals of Image Compression

- The goal of image compression is to reduce the amount of data required to represent a digital image.



Slide credit: Ashish Ghosh

Why do We Need Image Compression?

Standard definition (SD) television movie (raw data)

$$30\frac{\text{frames}}{\text{sec}}\times(720\times480)\frac{\text{pixels}}{\text{frame}}\times3\frac{\text{bytes}}{\text{pixel}}=31{,}104{,}000\text{bytes/sec}$$

A two-hour movie

$$31{,}104{,}000\frac{\text{bytes}}{\text{sec}}\times\left(60^2\right)\frac{\text{sec}}{\text{hour}}\times2\text{hours}\approx224\text{GB}$$

Need 27 8.5GB dual-layer DVDs!

High-definition (HD) television 1920x1080x24 bits/image!

Slide credit: Yan Tong

Why do We Need Image Compression?

Standard definition (SD) television movie (raw data)

$$30\frac{\text{frames}}{\text{sec}}\times(720\times480)\frac{\text{pixels}}{\text{frame}}\times\frac{24\text{bits}}{\text{pixel}}=248{,}832{,}000\text{bit/sec}>200\text{Mbit/sec}$$

WAN modems	Ethernet LAN	WiFi WLAN	Mobile data

• 1972: Acoustic coupler 300 baud
• 1977: 1200 baud Vadic and Bell 212A
• 1986: ISDN introduced with two 64 kbit/s channels (160 kbit/s gross bit rate)
• 1990: v.32bis modems: 2400 / 4800 / 9600 / 19200 bit/s
• 1994: v.34 modems with 28.8 kbit/s
• 1995: v.90 modems with 56 kbit/s downstreams, 33.6 kbit/s upstreams
• 1999: v.92 modems with 56 kbit/s downstreams, 48 kbit/s upstreams
• 1998: ADSL up to 8 Mbit/s,
• 2003: ADSL2 up to 12 Mbit/s
• 2005: ADSL2+ up to 24 Mbit/s
	• 1972: IEEE 802.3 Ethernet 2.94 Mbit/s
	• 1985: 10b2 10 Mbit/s coax thinwire
	• 1990: 10bT 10 Mbit/s
	• 1995: 100bT 100 Mbit/s (125 Mbit/s gross bit rate)
	• 1999: 1000bT (Gigabit) 1 Gbit/s (1.25 Gbit/s gross bit rate)
	• 2003: 10GBASE 10 Gbit/s
	
	http://en.wikipedia.org/wiki/Bit_rate
		WiFi WLANs
		• 1997: 802.11 2 Mbit/s
		• 1999: 802.11b 11 Mbit/s
		• 1999: 802.11a 54 Mbit/s (72 Mbit/s gross bit rate)
		• 2003: 802.11g 54 Mbit/s (72 Mbit/s gross bit rate)
		• 2005: 802.11g (proprietary) 108 Mbit/s
		• 2007: 802.11n 600 Mbit/s
			• 1G:
			  • 1981: NMT 1200 bit/s
			• 2G:
			  • 1991: GSM CSD and D-AMPS 14.4 kbit/s
			  • 2003: GSM EDGE 296 kbit/s down, 118.4 kbit/s up
			• 3G:
			  • 2001: UMTS-FDD (WCDMA) 384 kbit/s
			  • 2007: UMTS HSDPA 14.4 Mbit/s
			  • 2008: UMTS HSPA 14.4 Mbit/s down, 5.76 Mbit/s up
			  • 2009: HSPA+ (Without MIMO) 28 Mbit/s downstreams (56 Mbit/s with 2x2 MIMO), 22 Mbit/s upstreams
			  • 2010: CDMA2000 EV-DO Rev. B 14.7 Mbit/s downstreams
			• Pre-4G:
			  • 2007: Mobile WiMAX (IEEE 802.16e) 144 Mbit/s down, 35 Mbit/s up.
			  • 2009: LTE 100 Mbit/s downstreams (360 Mbit/s with MIMO 2x2), 50 Mbit/s upstreams
			See also Comparison of mobile phone standards

3.5Mbits/sec with MPEG-2 compression

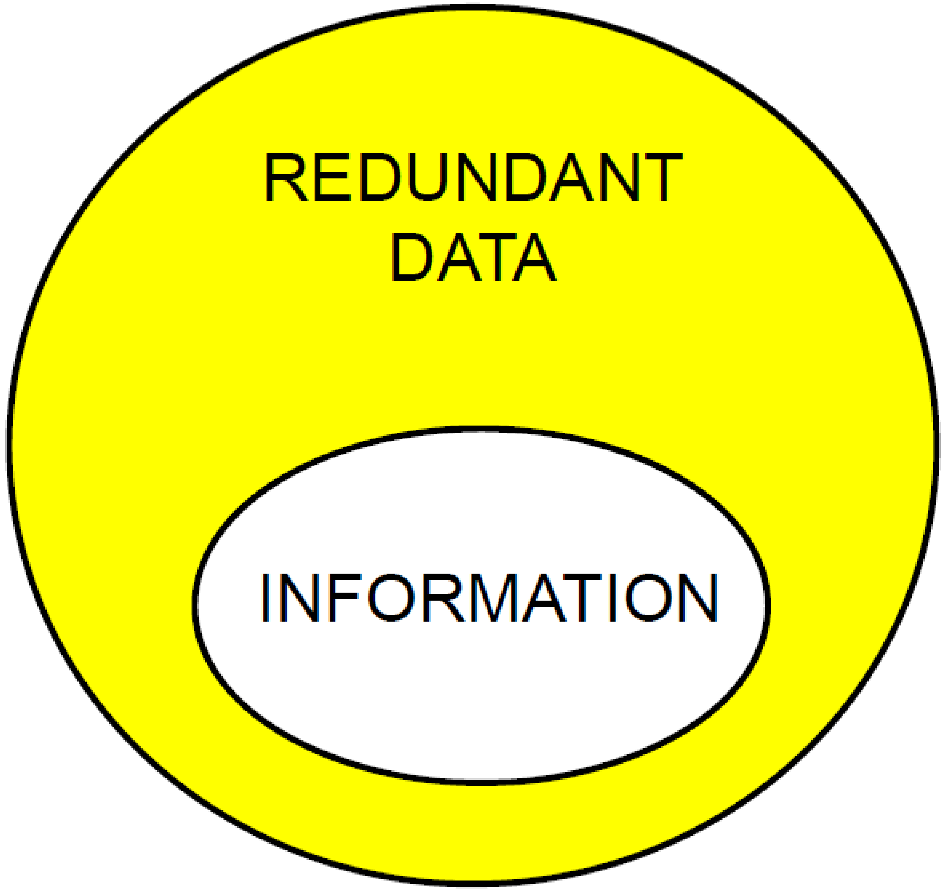
Slide credit: Yan Tong

Information vs Data

- The term data compression refers to the process of **reducing the amount of data** required to represent a given quantity of information
- Data $\neq$ Information
- Various amounts of data can be used to represent the same information
- Data might contain elements that provide no relevant information : *data redundancy*
- Data redundancy is a central issue in image compression.

Slide credit: Ashish Ghosh

Information vs Data



DATA = INFORMATION + REDUNDANT DATA

Slide credit: Ashish Ghosh

Data Redundancies

Data represent information – different ways

Representations that contain irrelevant or repeated information → contain redundant data

Two representations of the same information: b and b' bits, then the relative data redundancy

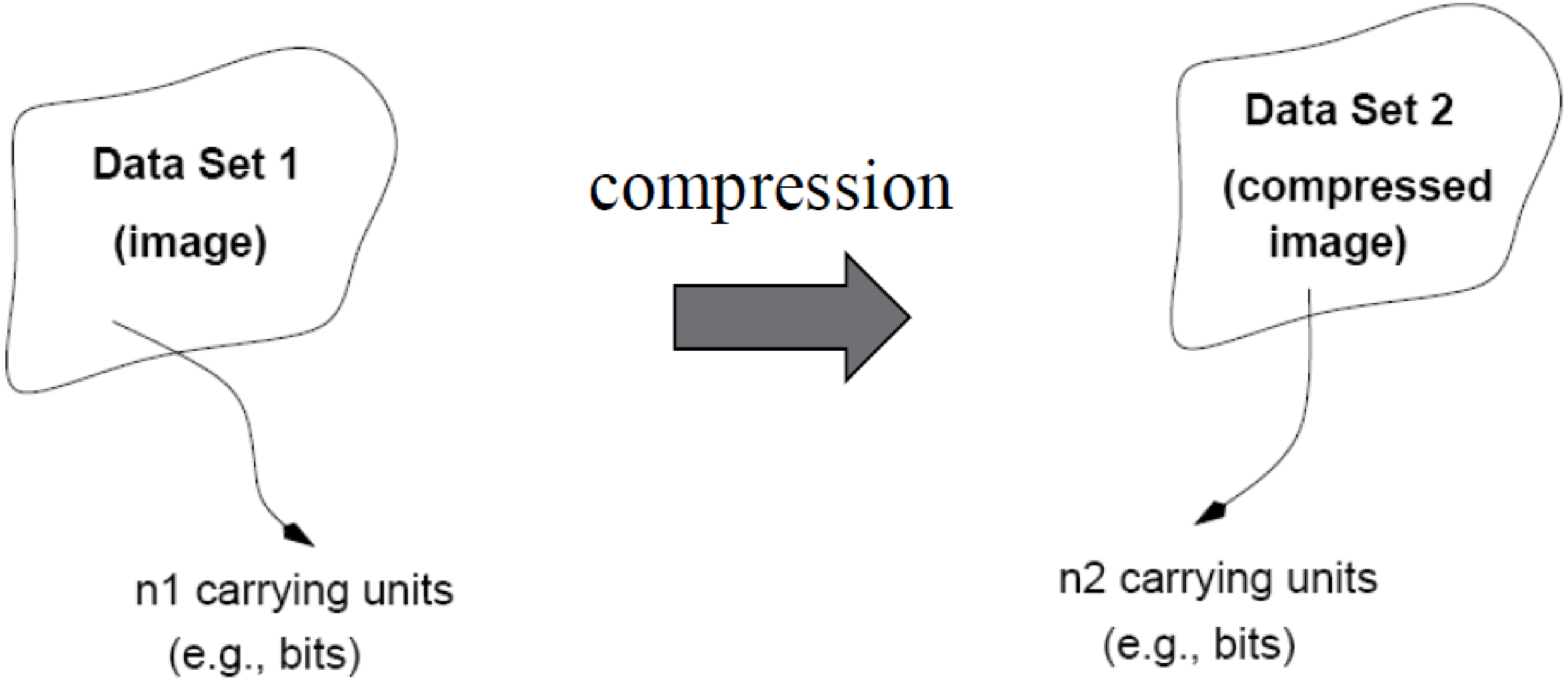
$$R = 1 - \frac{1}{C}, \quad \text{where}$$

$$C = \frac{b}{b'} \text{ is called the compression ratio}$$

In digital image processing, b is the # bits for the 2D array representation and b' is the compressed representation

Slide credit: Yan Tong

Definitions: Compression Ratio



Compression ratio: $C_R = \frac{n_1}{n_2}$

Slide credit: Ashish Ghosh

Definitions: Data Redundancy

Relative data redundancy: $R_D = 1 - \frac{1}{C_R}$

Example:

If $C_R = \frac{10}{1}$, then $R_D = 1 - \frac{1}{10} = 0.9$

(90% of the data in dataset 1 is redundant)

if $n_2 = n_1$, then C_R=1, R_D=0

if $n_2 \ll n_1$, then $C_R \to \infty$, $R_D \to 1$

Slide credit: Ashish Ghosh

Measuring Information

- What is the information content of a message/image?
- What is the minimum amount of data that is sufficient to describe completely an image without loss of information?

Modeling Information

- We assume that information generation is a probabilistic process.
- Idea: associate information with probability!

A random event E with probability $P(E)$ contains:

$$I(E) = log(\frac{1}{P(E)}) = -log(P(E)) \text{ units of information}$$

Note: I(E)=0 when P(E)=1

Slide credit: Ashish Ghosh

How much information does a pixel contain?

- Suppose that gray level values are generated by a random variable, then r_k contains:

$$I(r_k) = -log(P(r_k))$$ units of information!

Slide credit: Ashish Ghosh

How much information does a pixel contain?

- Average information content of an image:

$$E = \sum_{k=0}^{L-1} I(r_k)\,\Pr(r_k)$$

using $I(r_k) = -\,log(P(r_k))$

Entropy: $H = -\sum_{k=0}^{L-1} P(r_k) log(P(r_k))$ units/pixel (e.g., bits/pixel)

Slide credit: Ashish Ghosh

Entropy Estimation

- It is not easy to estimate H reliably!

image

21	21	21	95	169	243	243	243
21	21	21	95	169	243	243	243
21	21	21	95	169	243	243	243
21	21	21	95	169	243	243	243

Gray Level	*Count*	*Probability*
21	12	3/8
95	4	1/8
169	4	1/8
243	12	3/8

Slide credit: Ashish Ghosh

Entropy Estimation

- First order estimate of H:

$$H = -\sum_{k=0}^{3} P(r_k)log(P(r_k)) = 1.81 \text{ bits/pixel}$$

Total bits: 4 x 8 x 1.81 = 58 bits

Slide credit: Ashish Ghosh

Entropy Estimation

- Second order estimate of H:
  - Use relative frequencies of pixel blocks :

image

21	21	21	95	169	243	243	243
21	21	21	95	169	243	243	243
21	21	21	95	169	243	243	243
21	21	21	95	169	243	243	243

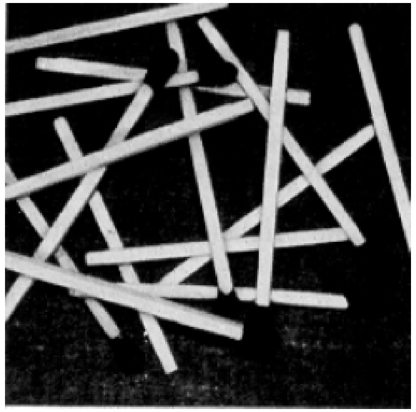
Gray Level Pair	*Count*	*Probability*
(21, 21)	8	1/4
(21, 95)	4	1/8
(95, 169)	4	1/8
(169, 243)	4	1/8
(243, 243)	8	1/4
(243, 21)	4	1/8

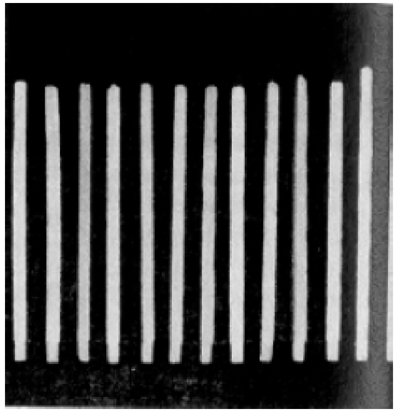
$$H = 2.5/2 = 1.25 \text{ bits/pixel}$$

Slide credit: Ashish Ghosh

Entropy Estimation

- The first-order estimate provides only a lower-bound on the compression that can be achieved.
- Differences between higher-order estimates of entropy and the first-order estimate indicate the presence of inter-pixel redundancy!.
- Inter-pixel redundancy implies that any pixel value can be reasonably predicted by its neighbors (i.e., correlated).





Slide credit: Ashish Ghosh

Redundancy

- Redundancy: $R = L_{avg} - H$

where: $L_{avg} = E(l(r_k)) = \sum_{k=0}^{L-1} l(r_k)P(r_k)$

and **$\mathbf{l(r_k)}$:** # of bits for r_k

<u>**Note**</u>**:** if $L_{avg} = H$, then $R=0$ (no redundancy)

Slide credit: Ashish Ghosh

Redundancy

Coding redundancy

- Code/code book is a system to represent information
- Code length is the number of symbols in each code word
- Do we really need 8 bits to represent a gray-level pixel?

Spatial and temporal redundancy

- Neighboring (spatially or temporally) pixels usually have similar intensities!
- Do we need to represent every pixel?

Irrelevant information

- Some image information can be ignored.

Slide credit: Yan Tong

Coding Redundancy

Histogram

$$p_r(r_k) = \frac{n_k}{MN}, \quad k = 0,1,2,\ldots, L-1$$

Average # bits required to represent each pixel

$$L_{\text{avg}} = \sum_{k=0}^{L-1} l(r_k) p_r(r_k)$$

Number of bits representing each intensity level

Total bits MNL_{avg}

r_k	$p_r(r_k)$	Code 1	$l_1(r_k)$	Code 2	$l_2(r_k)$
$r_{87} = 87$	0.25	01010111	8	01	2
$r_{128} = 128$	0.47	10000000	8	1	1
$r_{186} = 186$	0.25	11000100	8	000	3
$r_{255} = 255$	0.03	11111111	8	001	3
r_k for $k \neq 87, 128, 186, 255$	0	—	8	—	0

TABLE 8.1 Example of variable-length coding.

Fixed length Variable length

$$L_{avg} = 2 * 0.25 + 0.47 * 1 + 0.25 * 3 + 0.03 * 3 = 1.8$$

Slide credit: Yan Tong

Coding Redundancy

TABLE 8.1 Example of variable-length coding.

r_k	$p_r(r_k)$	Code 1	$l_1(r_k)$	Code 2	$l_2(r_k)$
$r_{87} = 87$	0.25	01010111	8	01	2
$r_{128} = 128$	0.47	10000000	8	1	1
$r_{186} = 186$	0.25	11000100	8	000	3
$r_{255} = 255$	0.03	11111111	8	001	3
r_k for $k \neq 87, 128, 186, 255$	0	—	8	—	0

C and R with variable length coding?

$$C = \frac{8}{1.81} = 4.42$$

$$R = 1 - \frac{1}{C} = 0.77$$

Slide credit: Yan Tong

Spatial and Temporal Redundancy

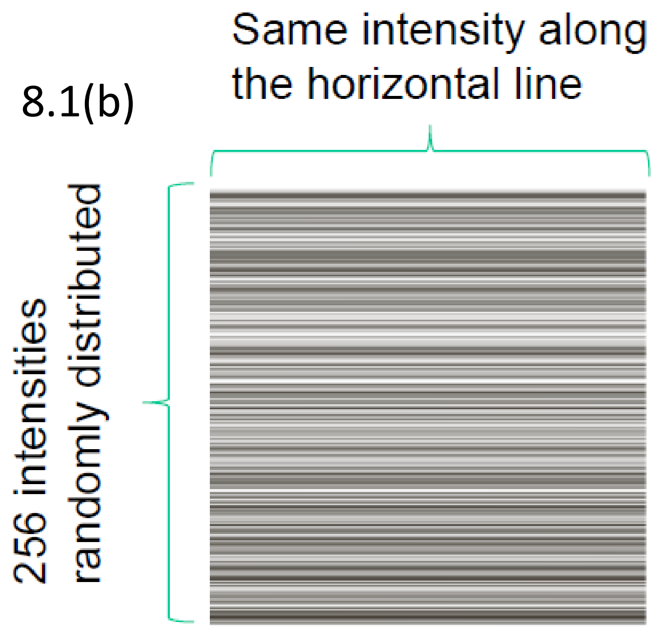
- Spatial redundancy
  - Neighboring pixels are not independent but correlated

- Temporal redundancy

Slide credit: Ashish Ghosh

Spatial and Temporal Redundancy

8.1(b)



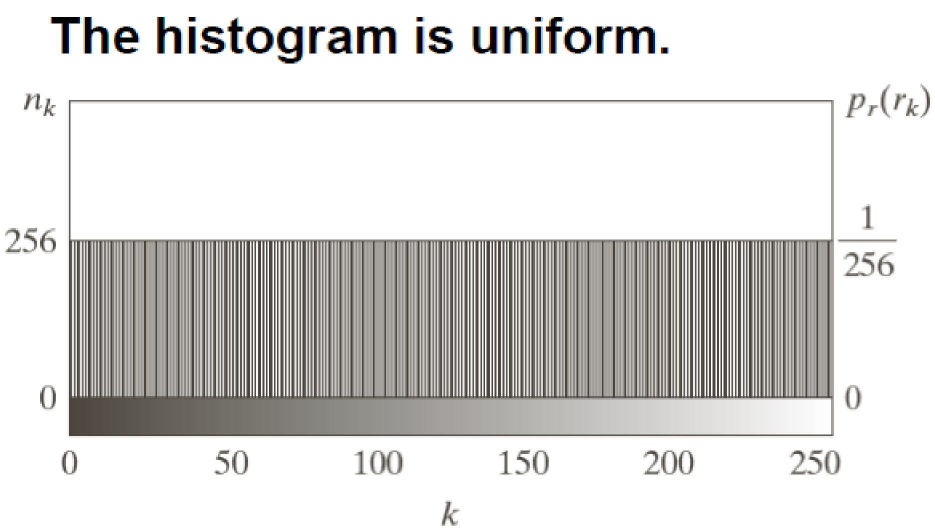


FIGURE 8.2 The intensity histogram of the image in Fig. 8.1(b).

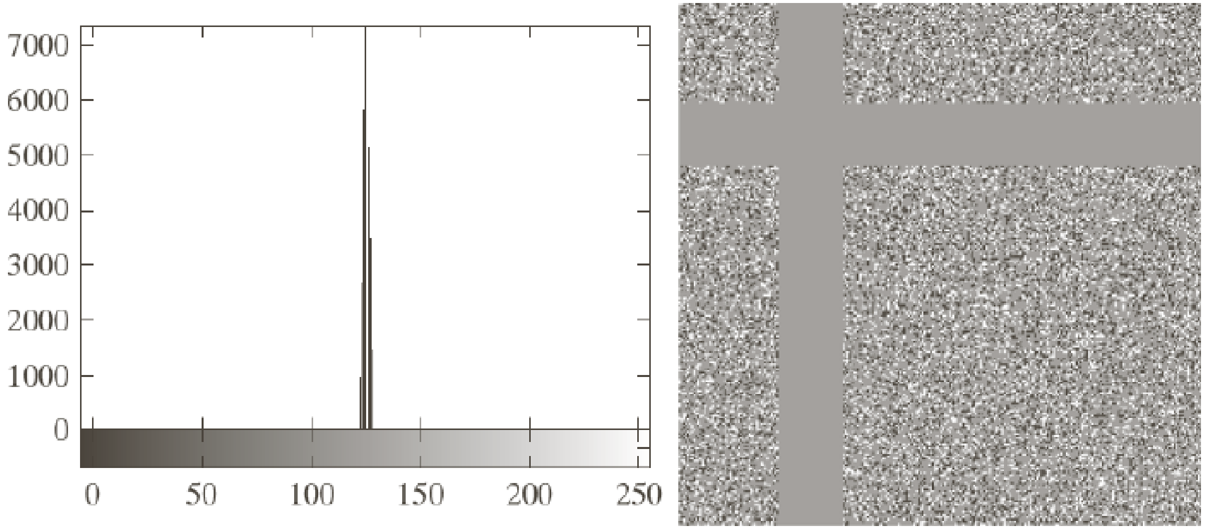
Compression by mapping:

- Run-length coding:
  - one word representing the intensity, and one word representing the length
  - C and R?
- Difference between two neighboring pixels

Slide credit: Yan Tong

Irrelevant Information

8.1(c)



a b

FIGURE 8.3
(a) Histogram of the image in Fig. 8.1(c) and (b) a histogram equalized version of the image.

Represented by a single byte

More details

The original 256 * 256 * 8 bit intensity array is reduced to a single byte, and the resulting compression is (256 * 256 * 8)/8 or 65,536:1.

Quantization – loss of quantitative information: irreversible operation

Slide credit: Yan Tong

Summary: Measuring Image Information

Minimum amount of data without losing information?

A random event E with probability $P(E)$ contains information

$$I = \log \frac{1}{P(E)} = -\log P(E)$$

Entropy (average information per image intensity)

$$H = -\sum_{k=0}^{L-1} p_r(r_k) \log_2 p_r(r_k)$$

Shannon's first theorem (noiseless coding theorem)

$$L_{\text{avg}} \geq H$$

and the low-bound *H* can be achieved by a coding method

Slide credit: Yan Tong

Fidelity Criteria – Quantify the Loss

objective fidelity criteria

Root mean square error

$$e_{\mathrm{rms}} = \left[\frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \left[\hat{f}(x,y) - f(x,y) \right]^2 \right]^{\frac{1}{2}}$$

Mean-square signal to noise ratio

$$SNR_{\mathrm{ms}} = \frac{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \hat{f}(x,y)^2}{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \left[\hat{f}(x,y) - f(x,y) \right]^2}$$

Slide credit: Yan Tong

Subjective Fidelity Criteria

Value	Rating	Description
1	Excellent	An image of extremely high quality, as good as you could desire.
2	Fine	An image of high quality, providing enjoyable viewing. Interference is not objectionable.
3	Passable	An image of acceptable quality. Interference is not objectionable.
4	Marginal	An image of poor quality; you wish you could improve it. Interference is somewhat objectionable.
5	Inferior	A very poor image, but you could watch it. Objectionable interference is definitely present.
6	Unusable	An image so bad that you could not watch it.

TABLE 8.2
Rating scale of the Television Allocations Study Organization. (Frendendall and Behrend.)

Slide credit: Yan Tong

Inconsistency between Objective and Subjective Fidelity Criteria

Objective: rms = 5.17 rms = 15.67 rms = 14.17

Subjective: Excellent Passable unusable

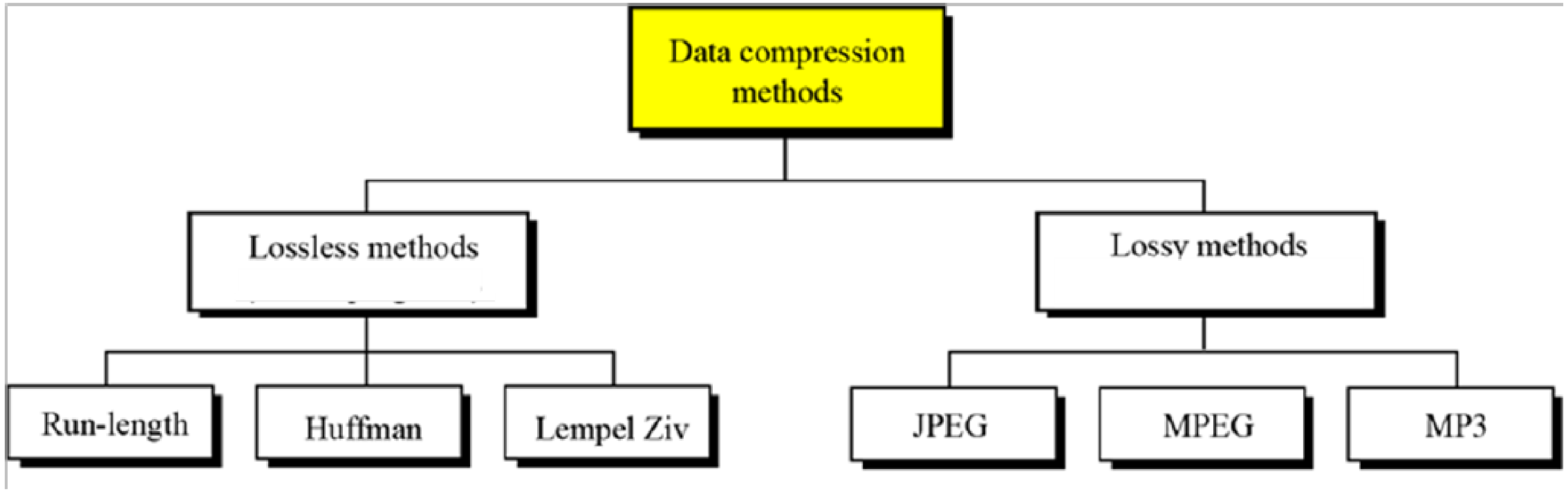
a b c

FIGURE 8.4 Three approximations of the image in Fig. 8.1(a).

Slide credit: Yan Tong

Data Compression

- Data compression aims at sending or storing a smaller number of bits.
- Although many methods are used for this purpose, in general these methods can be divided into two broad categories: lossless and lossy methods.



Slide credit: Ashish Ghosh

Lossless Compression

- In lossless data compression, the integrity of the data is preserved.
- The original data and the data after compression and decompression are exactly the same; as in these methods the compression and decompression algorithms are exact inverses of each other.
- No part of the data is lost in the process.
- Redundant data is removed in compression and added during decompression.
- Lossless compression methods are normally used when we cannot afford to lose any data.

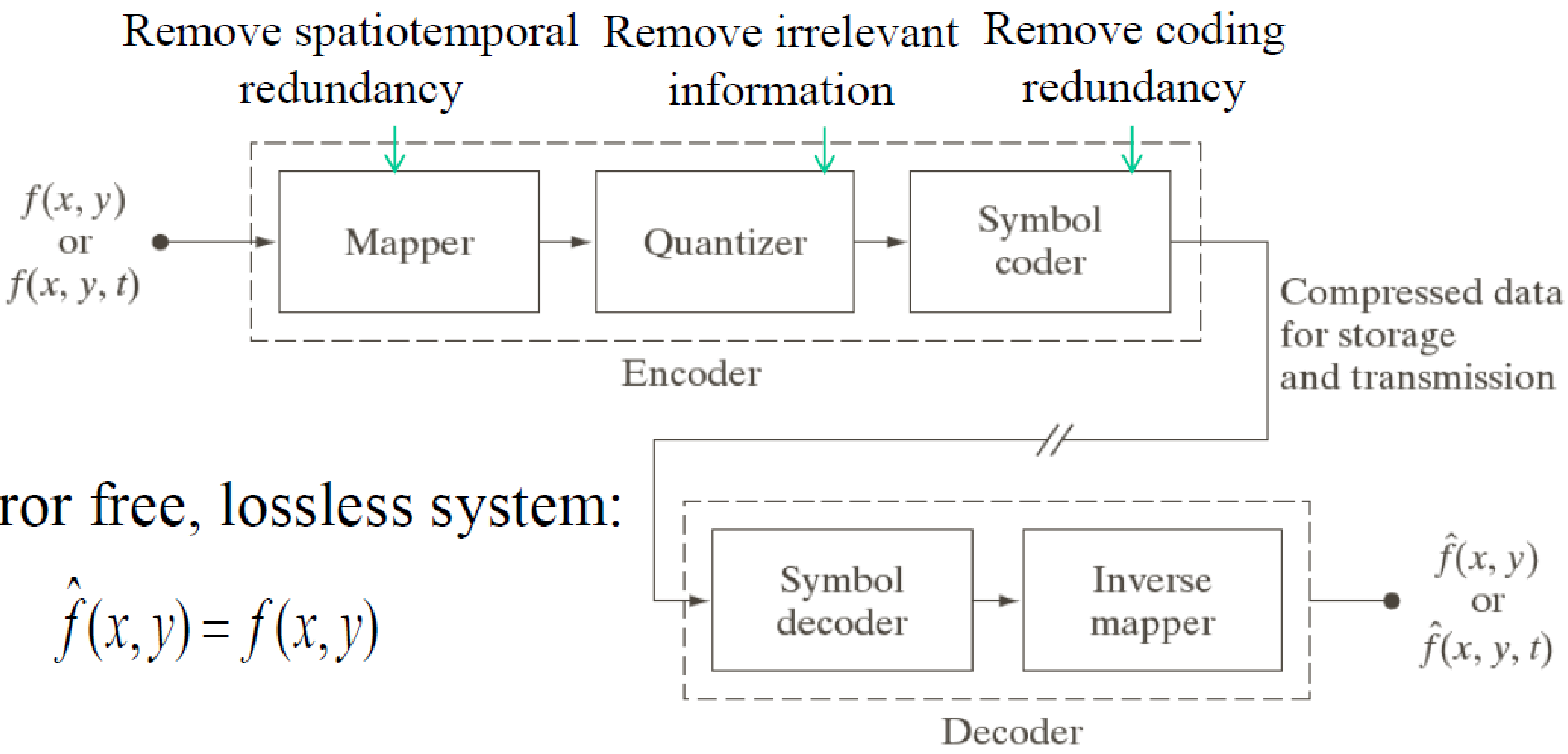
Slide credit: Ashish Ghosh

Lossy Compression

- Our eyes and ears cannot distinguish subtle changes.
- In such cases, we can use a lossy data compression method.
- These methods are cheaper—they take less time and space when it comes to sending millions of bits per second for images and video.
- Several methods have been developed using lossy compression techniques. JPEG (Joint Photographic Experts Group) encoding is used to compress pictures and graphics, MPEG (Moving Picture Experts Group) encoding is used to compress video, and MP3 (MPEG audio layer 3) for audio compression.

Slide credit: Ashish Ghosh

Image-Compression Models



Error free, lossless system:

$$\hat{f}(x,y) = f(x,y)$$

Lossy system:

$$\hat{f}(x,y) \neq f(x,y)$$

FIGURE 8.5 Functional block diagram of a general image compression system.

Slide credit: Yan Tong

Image Formats, Containers and Compression Standards

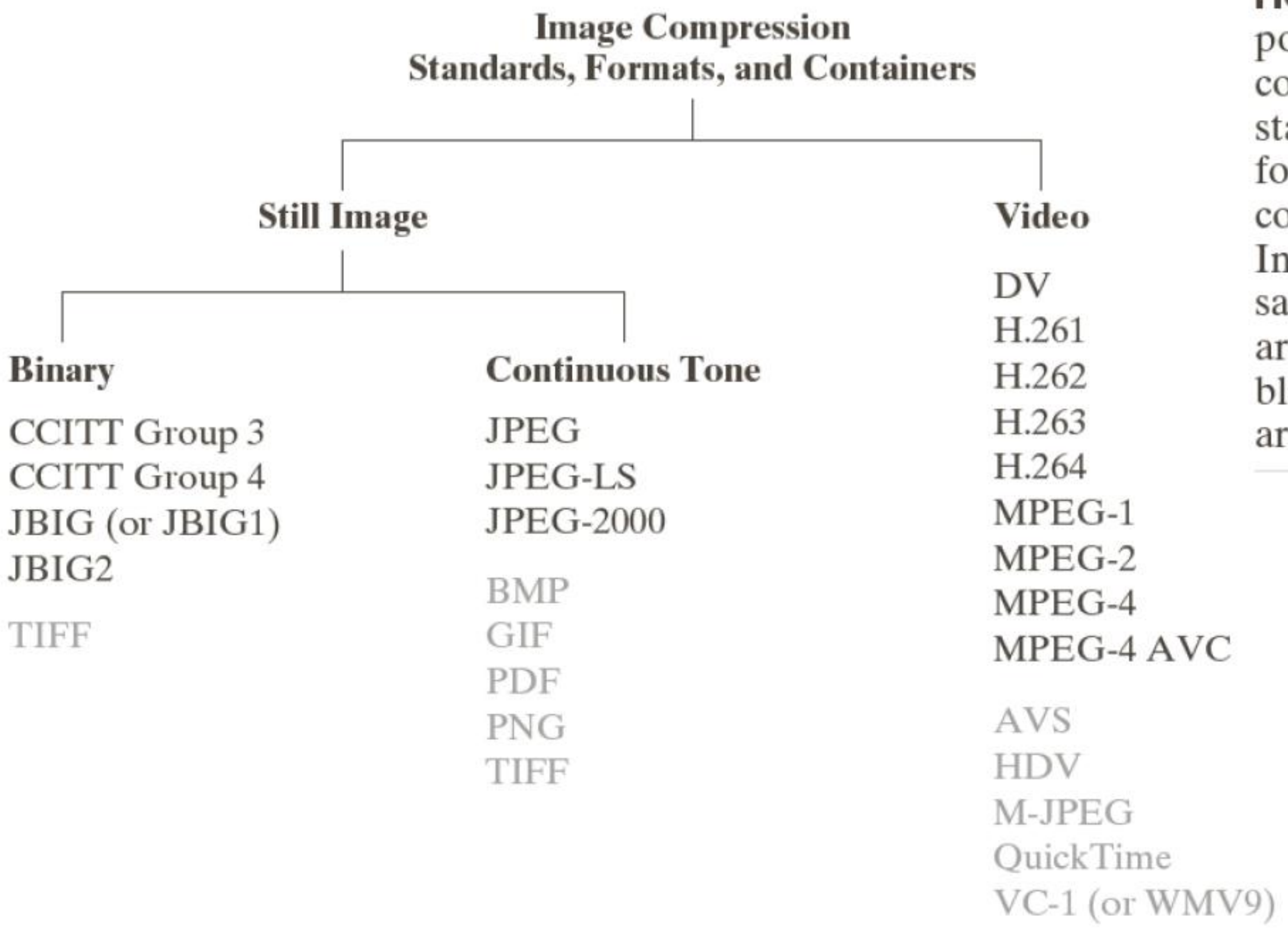


FIGURE 8.6 Some popular image compression standards, file formats, and containers. Internationally sanctioned entries are shown in black; all others are grayed.

Slide credit: Yan Tong

Some Basic Compression Methods – Huffman Coding (Block Code)

- Huffman codes can be used to compress information
  - Like WinZip – although WinZip doesn't use the Huffman algorithm
  - JPEGs do use Huffman as part of their compression process
- The basic idea is that instead of storing each character in a file as an 8-bit ASCII value, we will instead store the more frequently occurring characters using fewer bits and less frequently occurring characters using more bits
  - On average this should decrease the file size (usually ½)

Slide credit: Ashish Ghosh

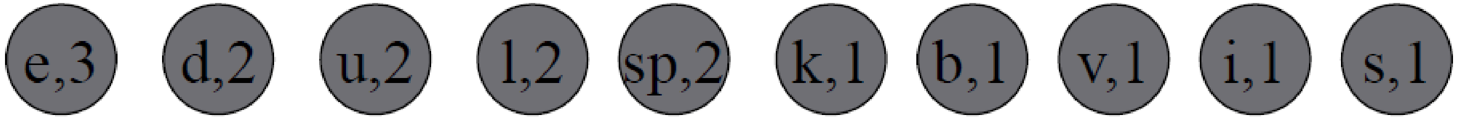
Some Basic Compression Methods – Huffman Coding (Block Code)

- As an example, lets take the string:

  "duke blue devils"

- We first find the frequency count of the characters:
  - e:3, d:2, u:2, l:2, space:2, k:1, b:1, v:1, i:1, s:1

- Next we use a Greedy algorithm to build up a Huffman Tree
  - We start with nodes for each character

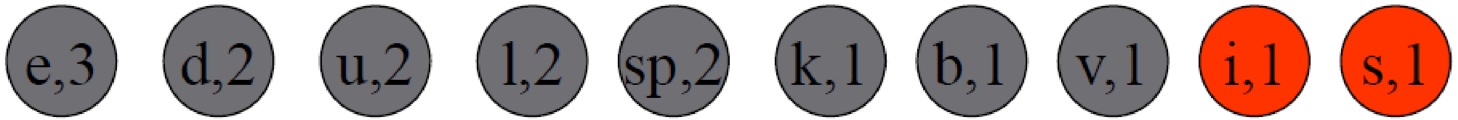


Slide credit: Ashish Ghosh

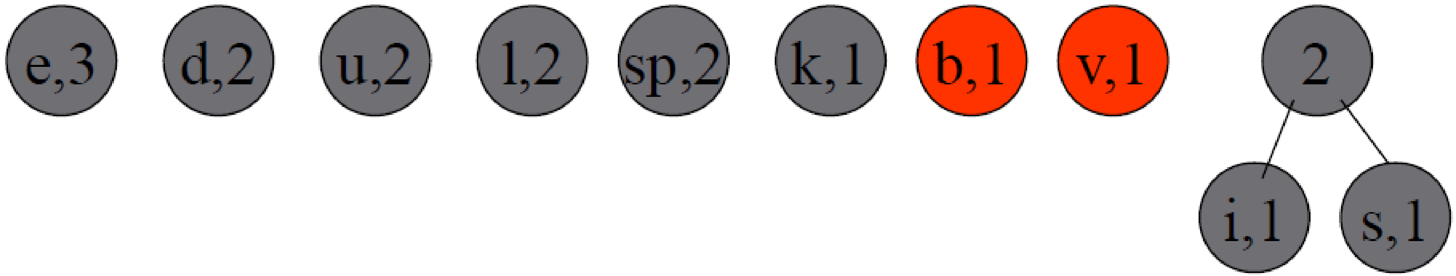
Some Basic Compression Methods – Huffman Coding (Block Code)

- We then pick the nodes with the smallest frequency and combine them together to form a new node
  - The selection of these nodes is the Greedy part
- The two selected nodes are removed from the set, but replaced by the combined node
- This continues until we have only 1 node left in the set

Some Basic Compression Methods – Huffman Coding (Block Code)

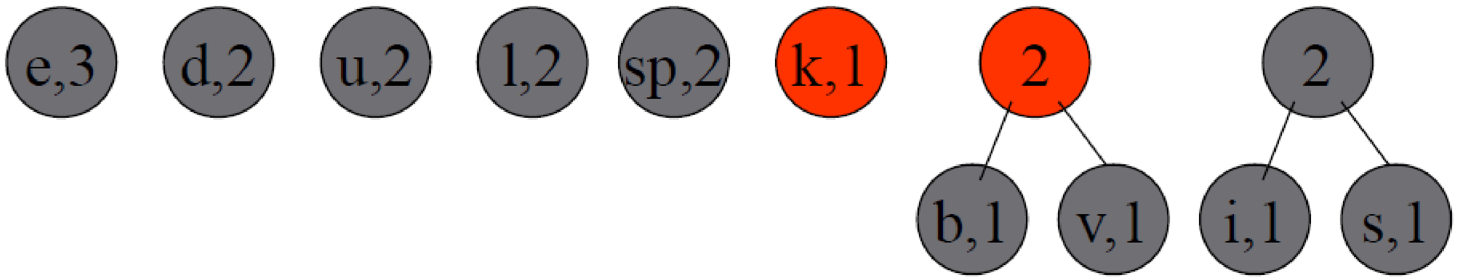


Some Basic Compression Methods – Huffman Coding (Block Code)



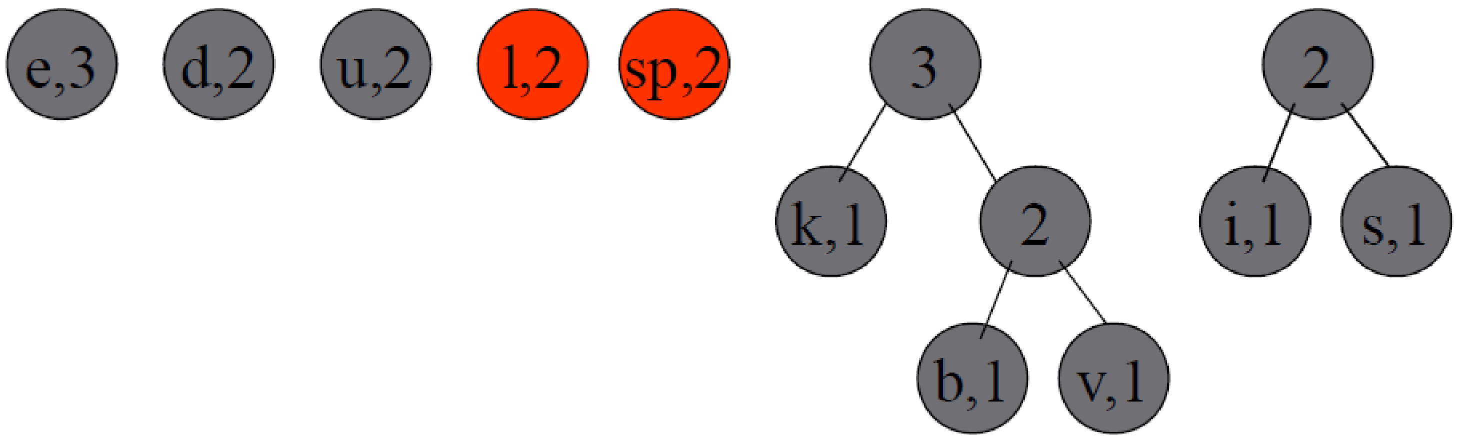
Slide credit: Ashish Ghosh

Some Basic Compression Methods – Huffman Coding (Block Code)



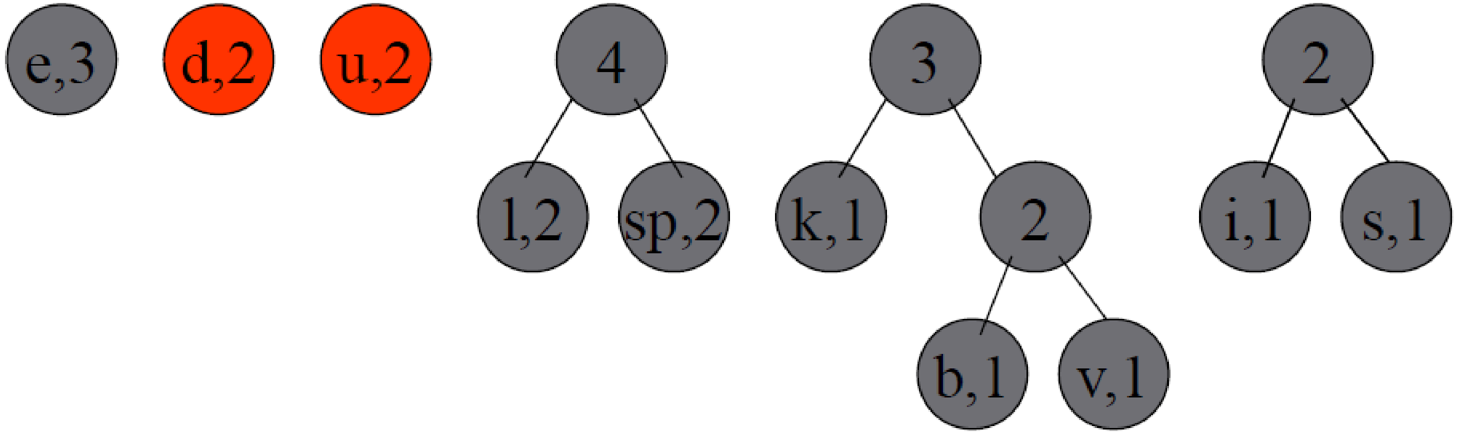
Slide credit: Ashish Ghosh

Some Basic Compression Methods – Huffman Coding (Block Code)



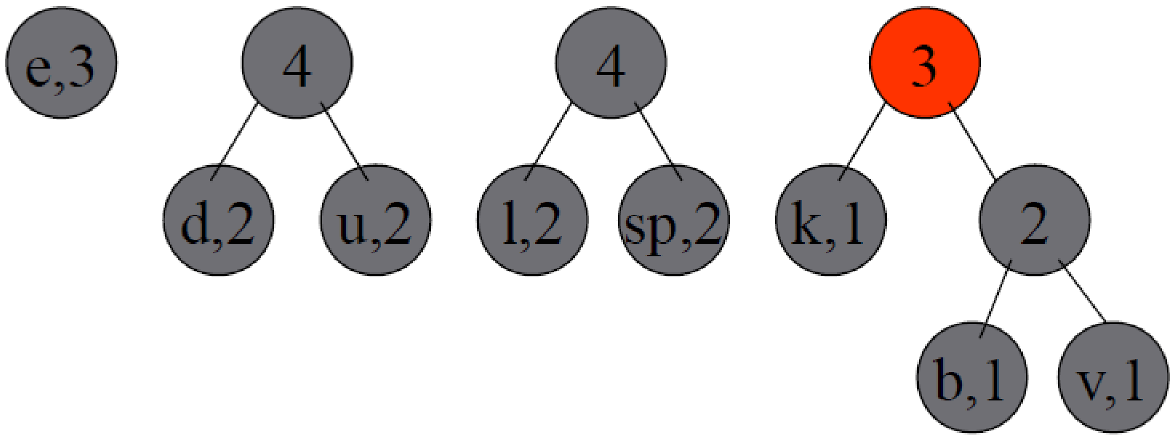
Slide credit: Ashish Ghosh

Some Basic Compression Methods – Huffman Coding (Block Code)



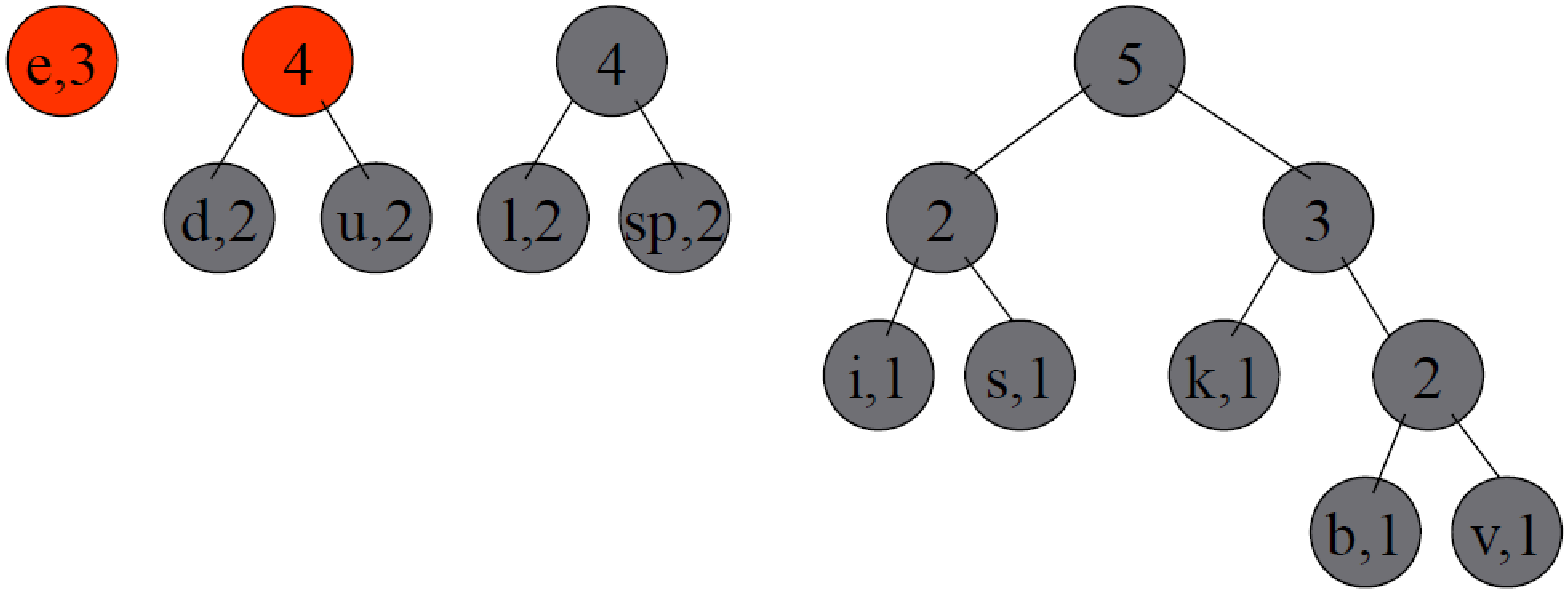
Slide credit: Ashish Ghosh

Some Basic Compression Methods – Huffman Coding (Block Code)



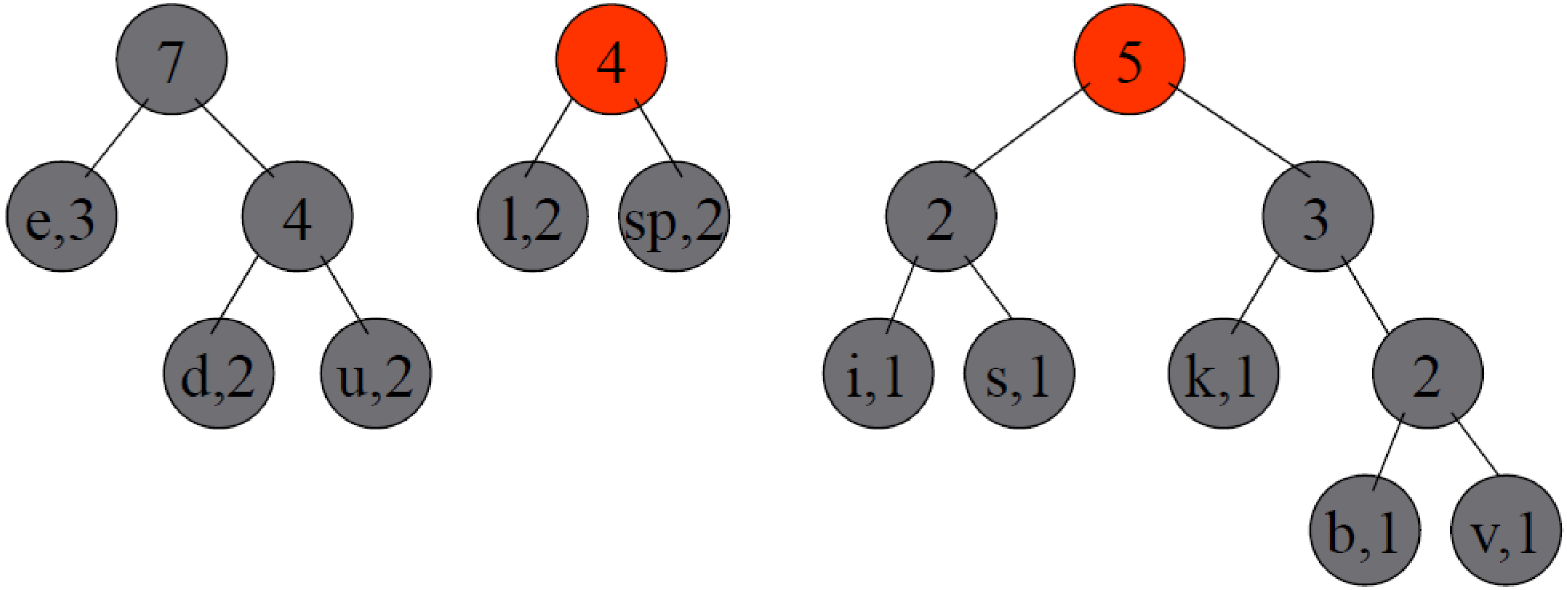
Slide credit: Ashish Ghosh

Some Basic Compression Methods – Huffman Coding (Block Code)



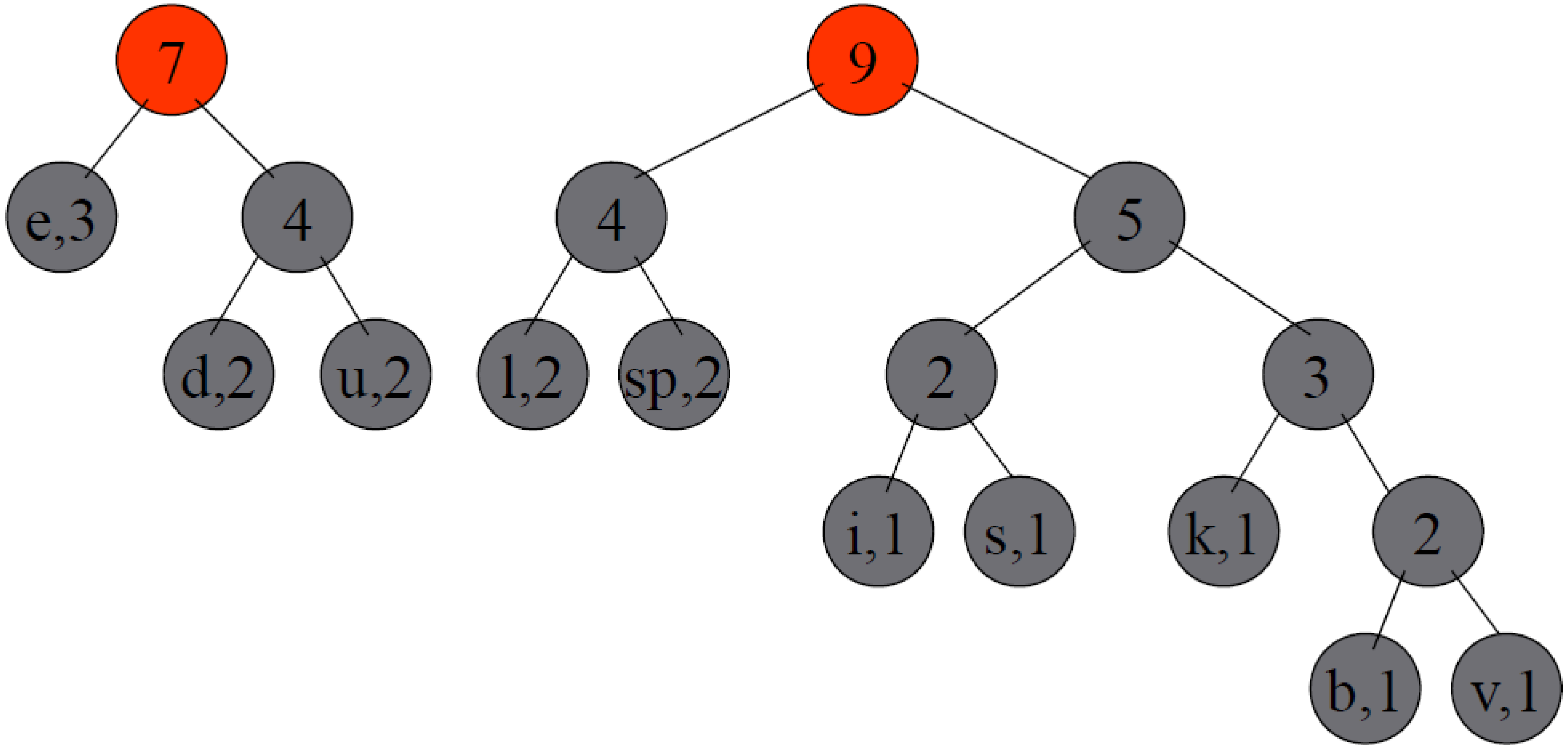
Slide credit: Ashish Ghosh

Some Basic Compression Methods – Huffman Coding (Block Code)



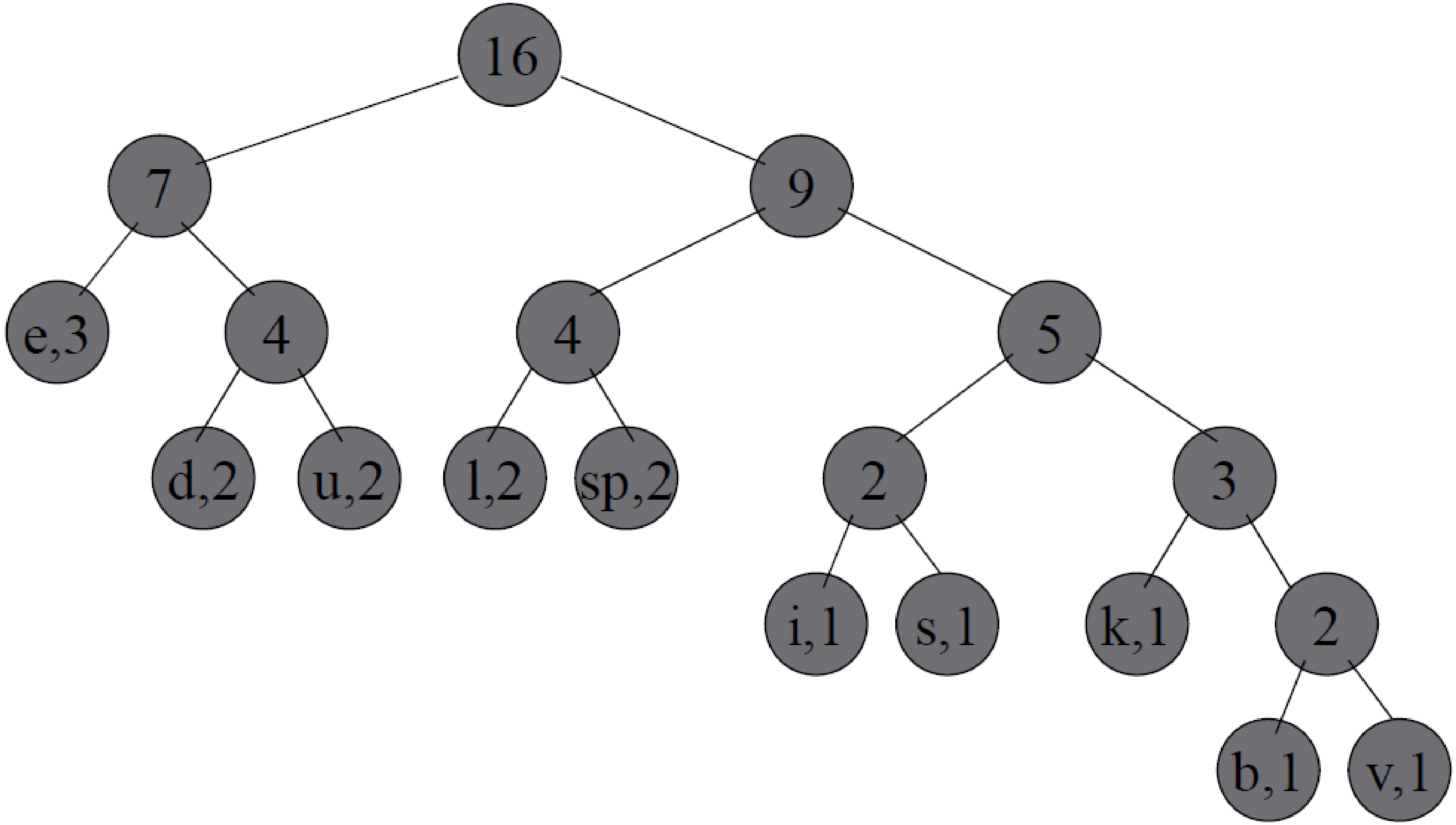
Slide credit: Ashish Ghosh

Some Basic Compression Methods – Huffman Coding (Block Code)



Slide credit: Ashish Ghosh

Some Basic Compression Methods – Huffman Coding (Block Code)



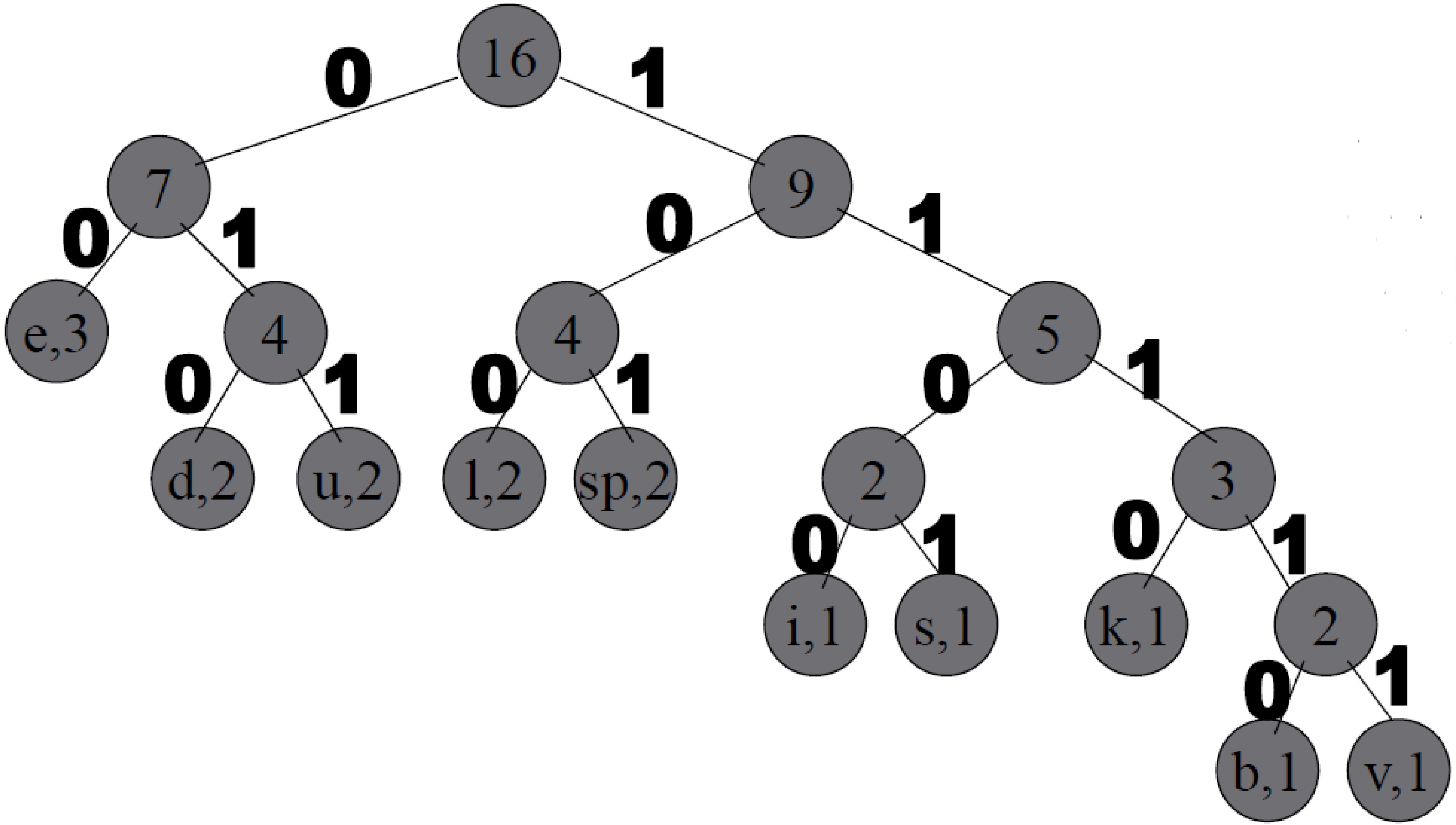
Slide credit: Ashish Ghosh

Some Basic Compression Methods – Huffman Coding (Block Code)

- Now we assign codes to the tree by placing a 0 on every left branch and a 1 on every right branch
- A traversal of the tree from root to leaf gives the Huffman code for that particular leaf character
- Note that no code is the prefix of another code

Slide credit: Ashish Ghosh

Some Basic Compression Methods – Huffman Coding (Block Code)



e	00
d	010
u	011
l	100
sp	101
i	1100
s	1101
k	1110
b	11110
v	11111

Slide credit: Ashish Ghosh

Some Basic Compression Methods – Huffman Coding (Block Code)

- These codes are then used to encode the string
- Thus, "duke blue devils" turns into:

  010 011 1110 00 101 11110 100 011 00 101 010 00
  11111 1100 100 1101

- When grouped into 8-bit bytes:

  01001111 10001011 11101000 11001010 10001111
  11100100 1101xxxx

- Thus it takes 7 bytes of space compared to 16 characters * 1 byte/char

  = 16 bytes uncompressed

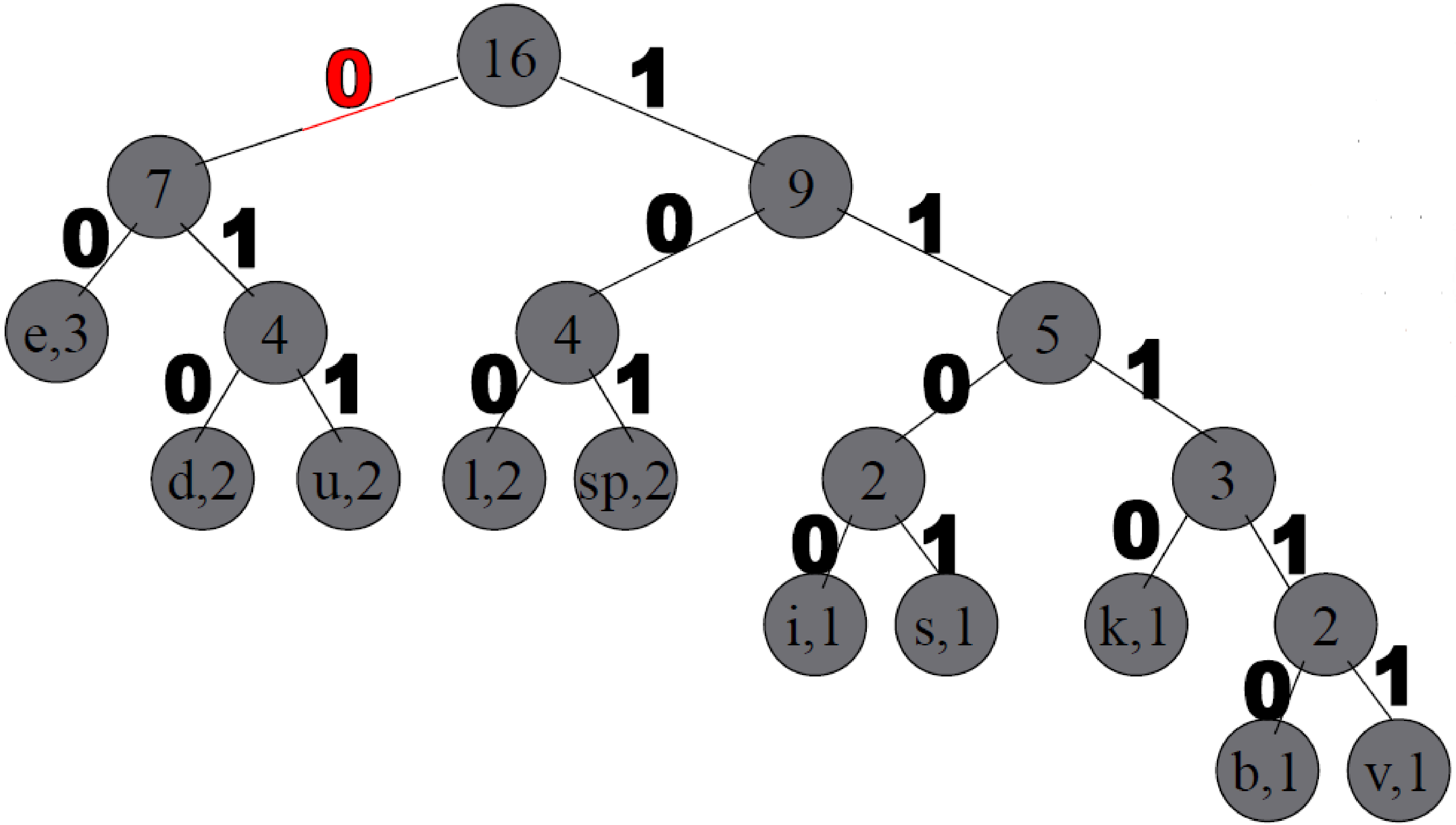
Slide credit: Ashish Ghosh

Some Basic Compression Methods – Huffman Coding (Block Code)

- Uncompressing works by reading in the file bit by bit
  - Start at the root of the tree
  - If a 0 is read, head left
  - If a 1 is read, head right
  - When a leaf is reached decode that character and start over again at the root of the tree
- Thus, we need to save Huffman table information as a header in the compressed file
  - Doesn't add a significant amount of size to the file for large files (which are the ones you want to compress anyway)
  - Or we could use a fixed universal set of codes/freqencies

Slide credit: Ashish Ghosh

Some Basic Compression Methods – Huffman Coding (Block Code)

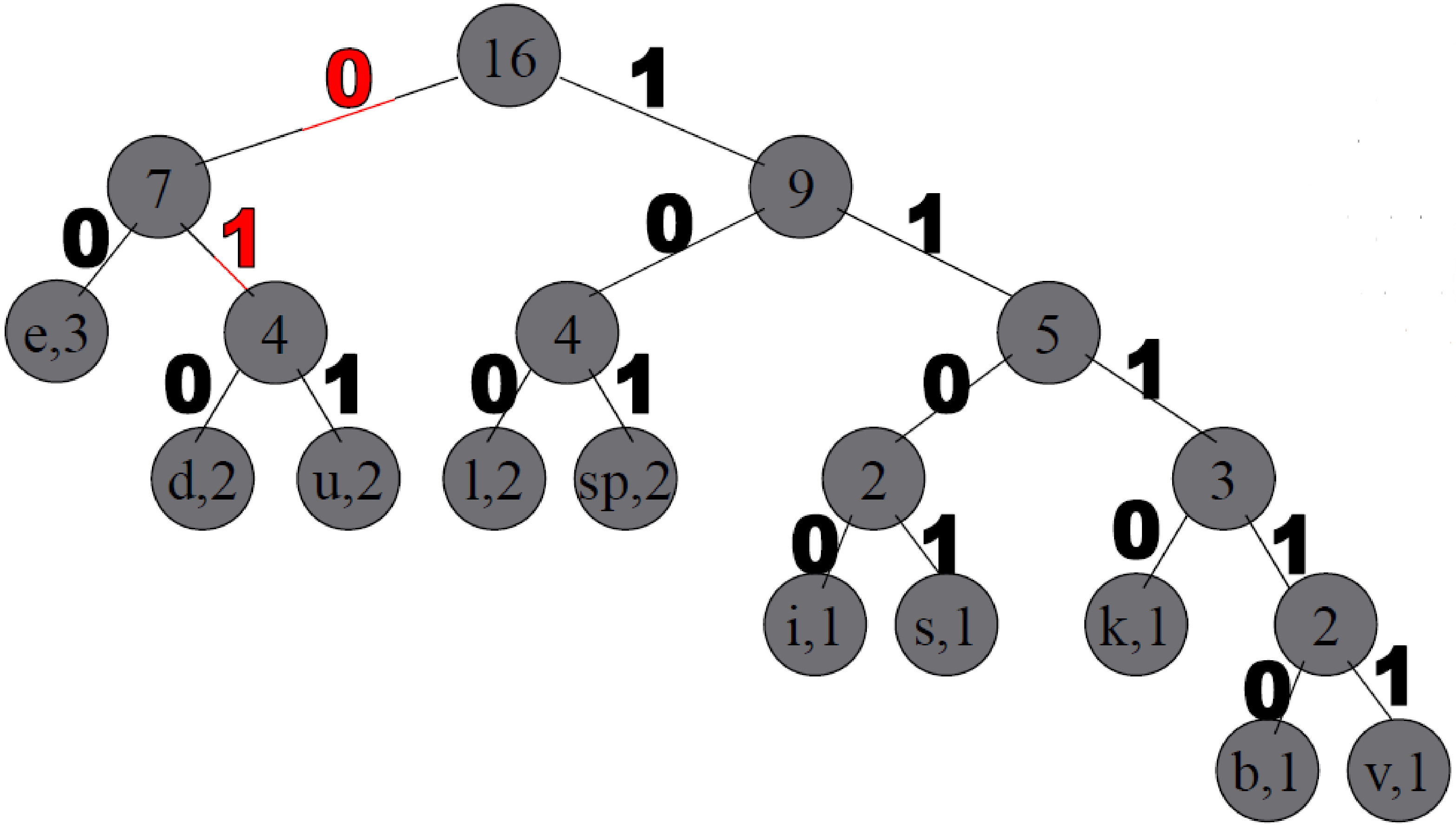


e	00
d	010
u	011
l	100
sp	101
i	1100
s	1101
k	1110
b	11110
v	11111

010 011 1110 00 101 11110 100 011 00 101 010 00 11111 1100 100 1101

Slide credit: Ashish Ghosh

Some Basic Compression Methods – Huffman Coding (Block Code)

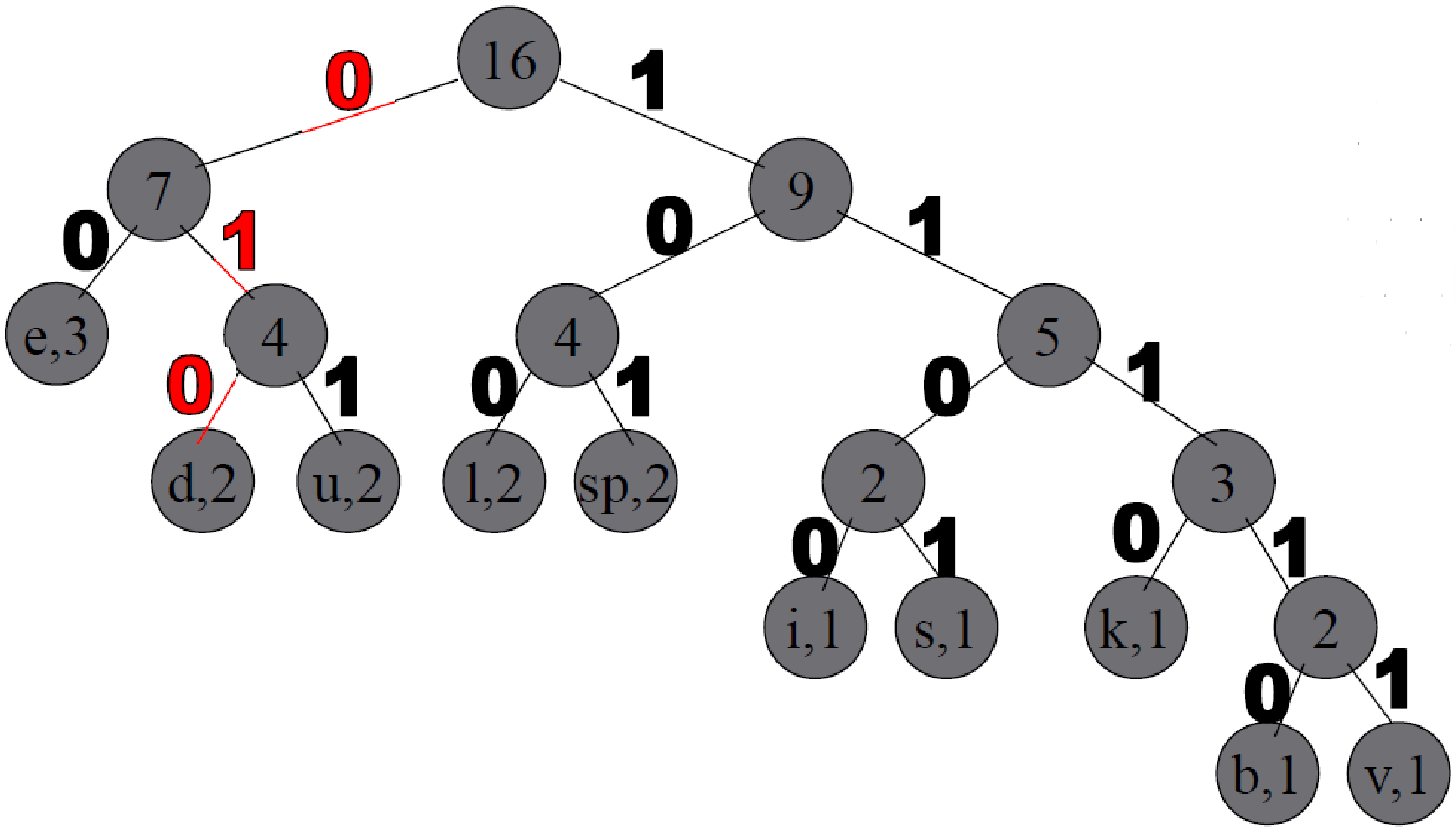


e	00
d	010
u	011
l	100
sp	101
i	1100
s	1101
k	1110
b	11110
v	11111

010 011 1110 00 101 11110 100 011 00 101 010 00 11111 1100 100 1101

Slide credit: Ashish Ghosh

Some Basic Compression Methods – Huffman Coding (Block Code)

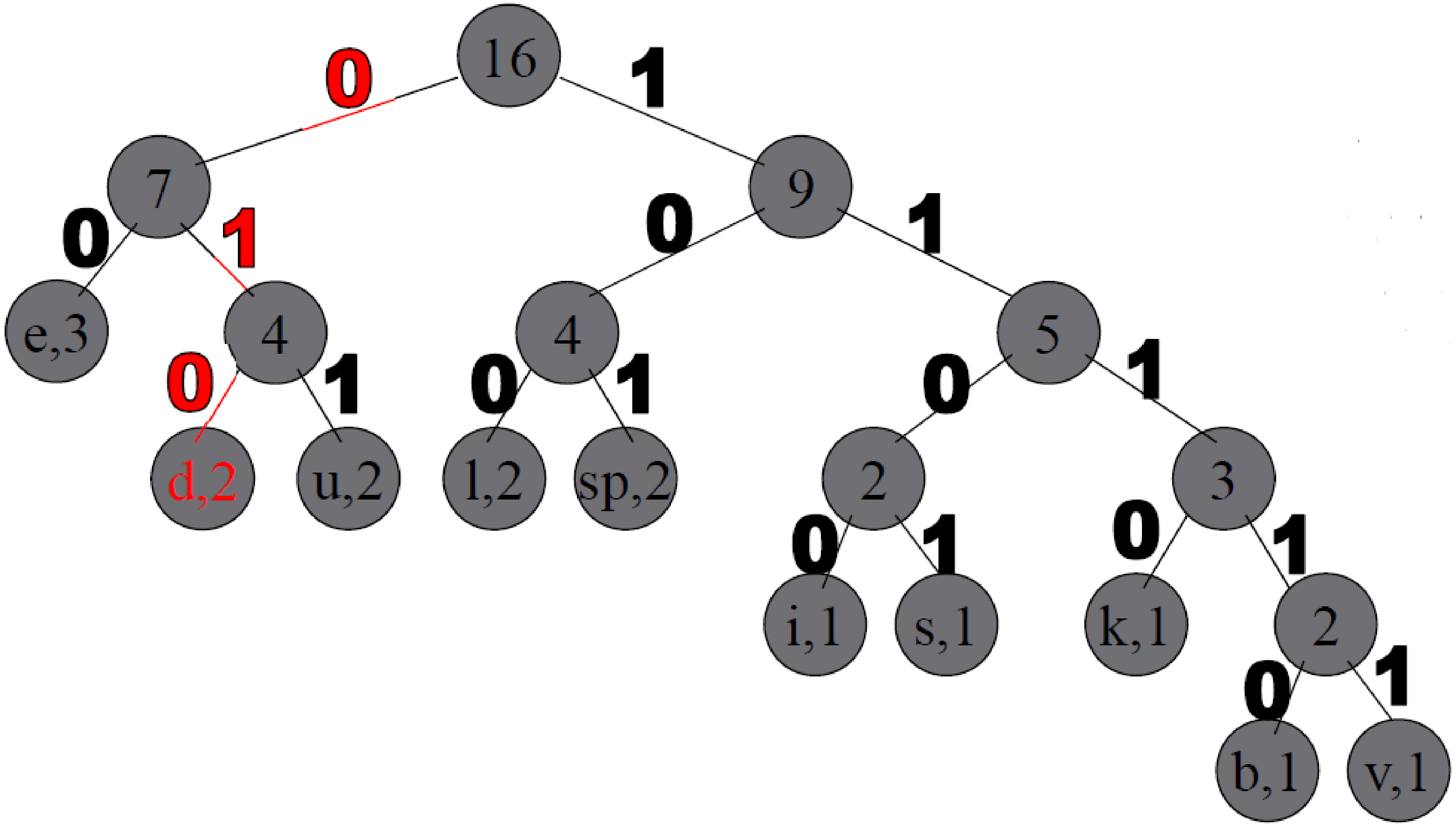


e	00
d	010
u	011
l	100
sp	101
i	1100
s	1101
k	1110
b	11110
v	11111

010 011 1110 00 101 11110 100 011 00 101 010 00 11111 1100 100 1101

Slide credit: Ashish Ghosh

Some Basic Compression Methods – Huffman Coding (Block Code)

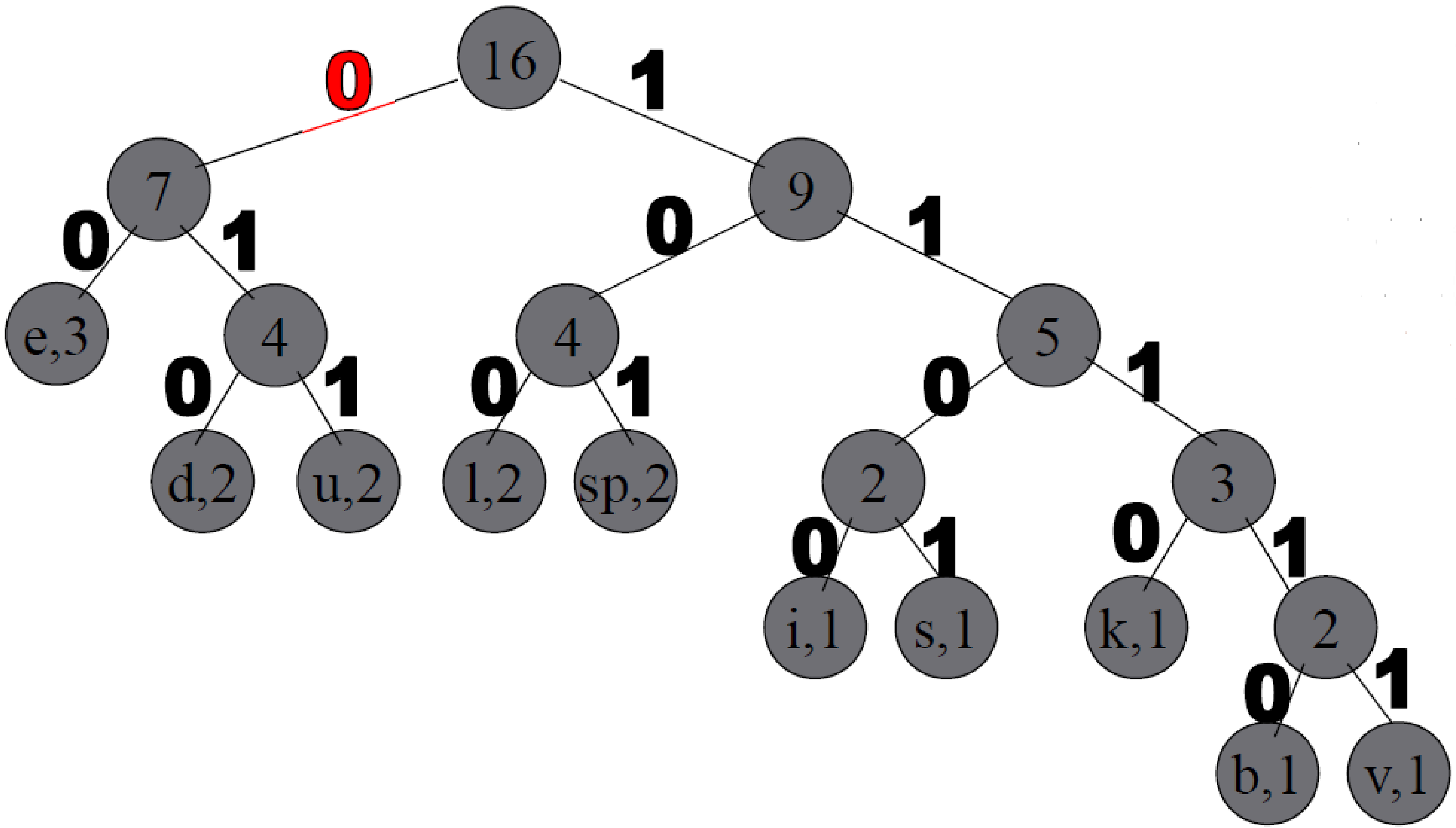


e	00
d	010
u	011
l	100
sp	101
i	1100
s	1101
k	1110
b	11110
v	11111

d 011 1110 00 101 11110 100 011 00 101 010 00 11111 1100 100 1101

Slide credit: Ashish Ghosh

Some Basic Compression Methods – Huffman Coding (Block Code)

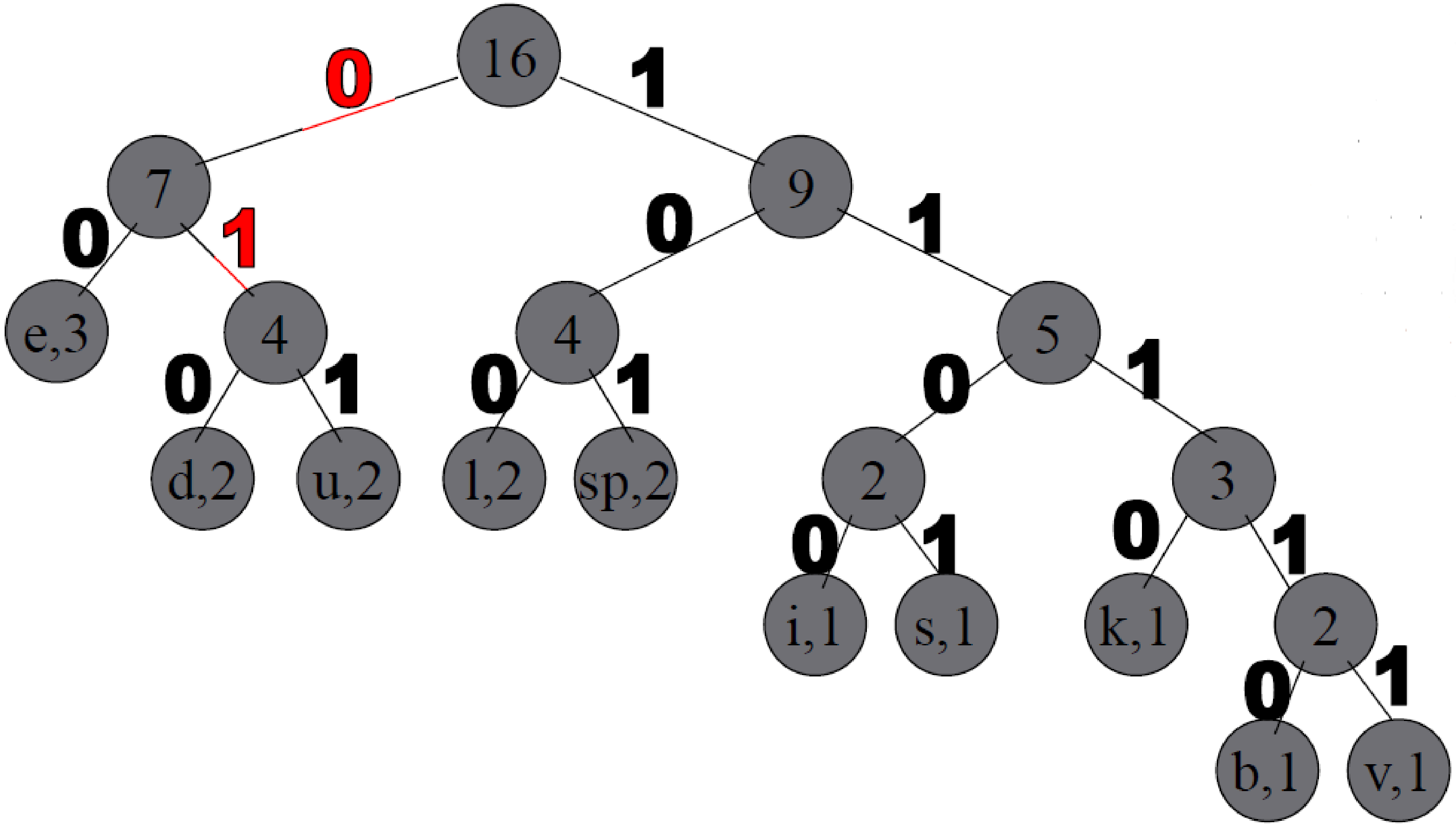


e	00
d	010
u	011
l	100
sp	101
i	1100
s	1101
k	1110
b	11110
v	11111

d 011 1110 00 101 11110 100 011 00 101 010 00 11111 1100 100 1101

Slide credit: Ashish Ghosh

Some Basic Compression Methods – Huffman Coding (Block Code)

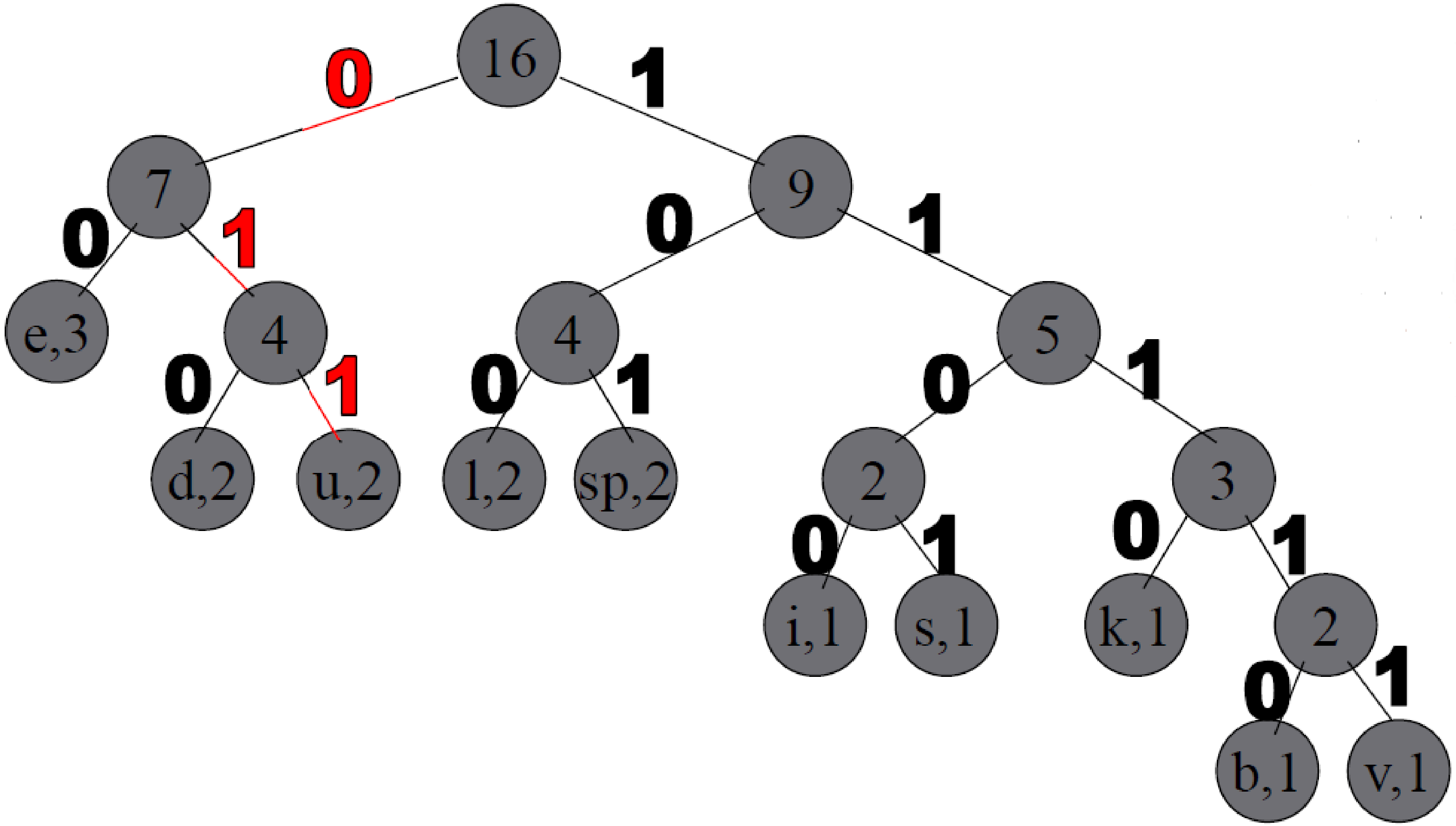


e	00
d	010
u	011
l	100
sp	101
i	1100
s	1101
k	1110
b	11110
v	11111

d 011 1110 00 101 11110 100 011 00 101 010 00 11111 1100 100 1101

Slide credit: Ashish Ghosh

Some Basic Compression Methods – Huffman Coding (Block Code)

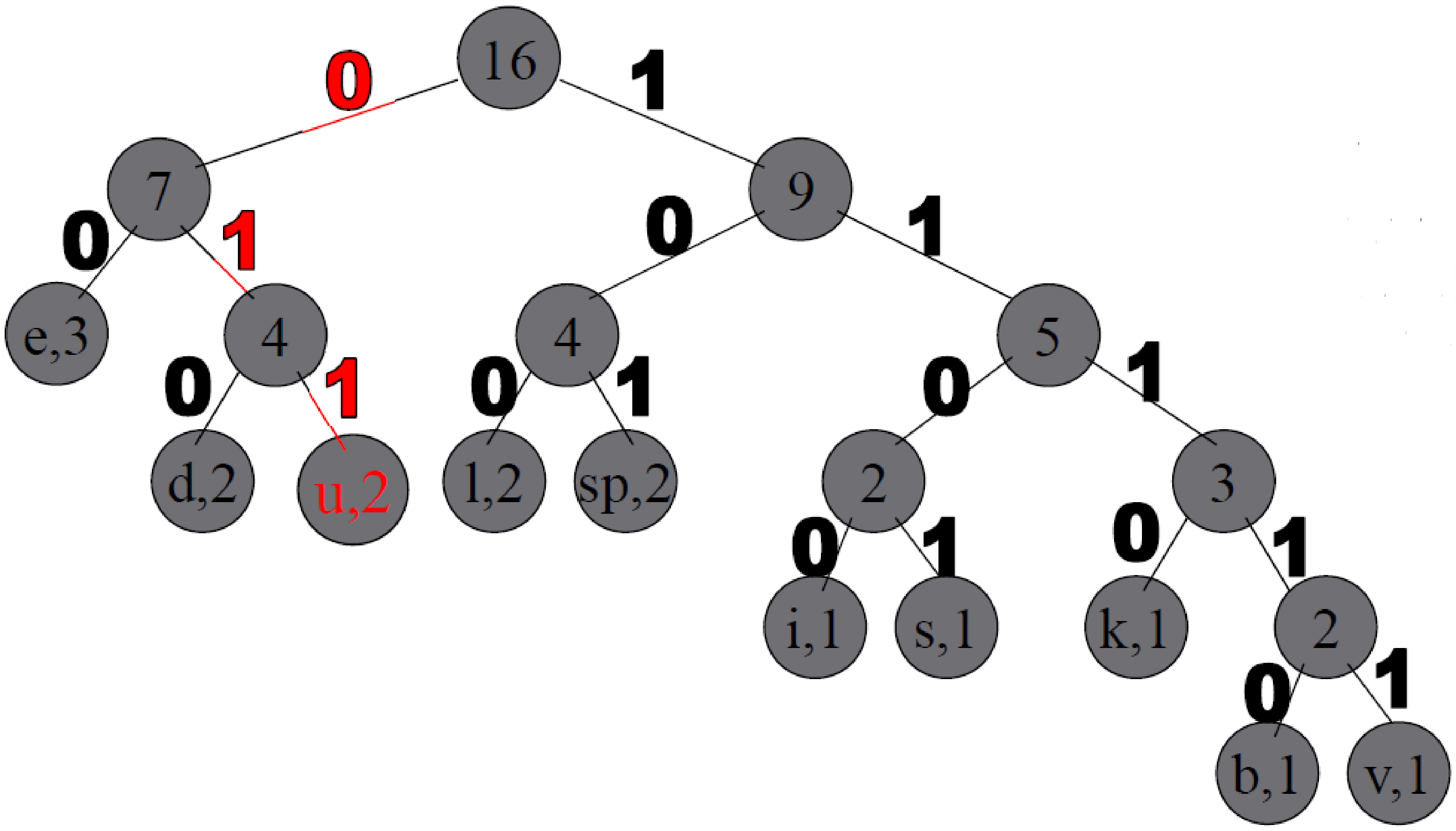


e	00
d	010
u	011
l	100
sp	101
i	1100
s	1101
k	1110
b	11110
v	11111

d 011 1110 00 101 11110 100 011 00 101 010 00 11111 1100 100 1101

Slide credit: Ashish Ghosh

Some Basic Compression Methods – Huffman Coding (Block Code)

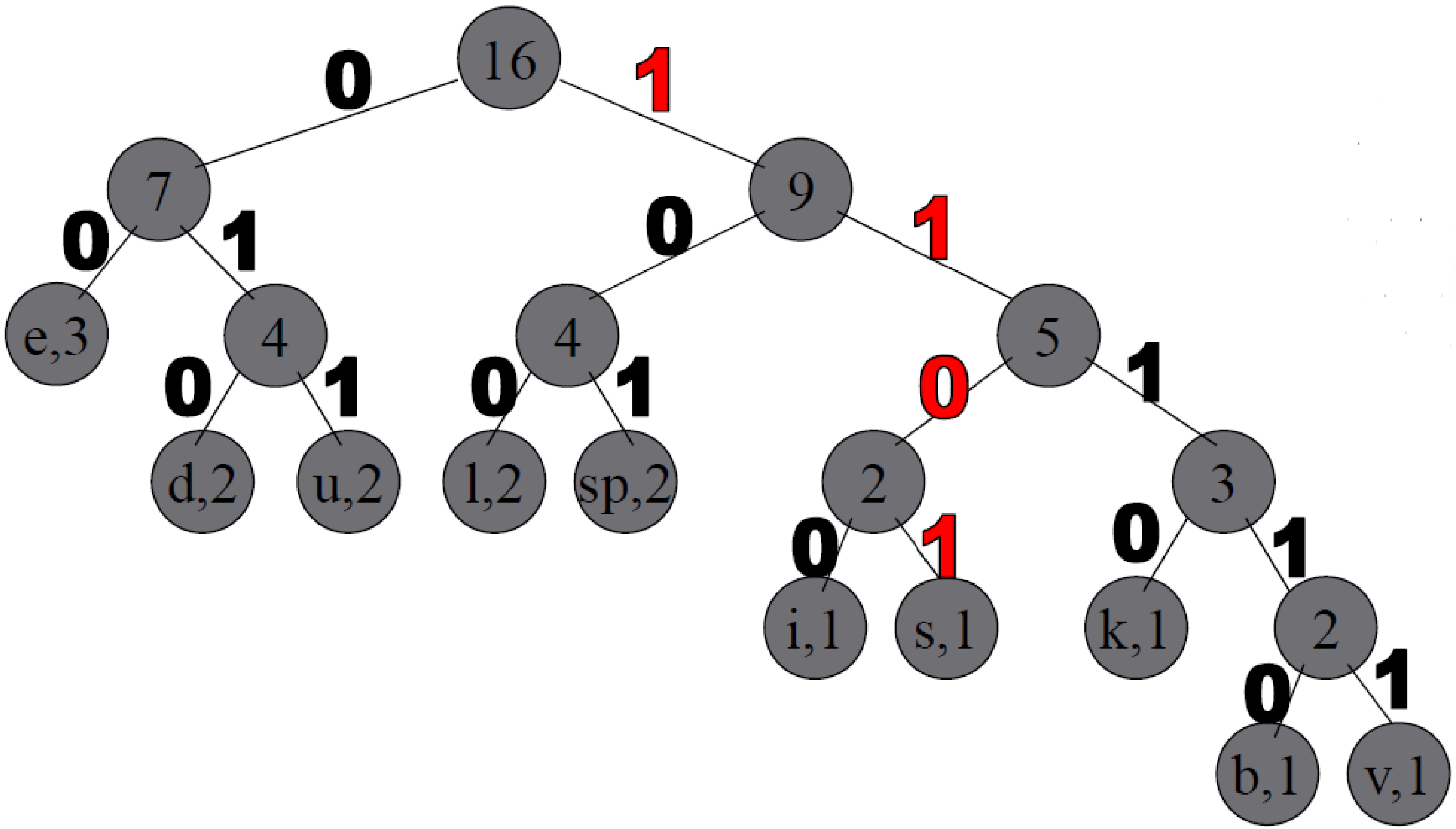


e	00
d	010
u	011
l	100
sp	101
i	1100
s	1101
k	1110
b	11110
v	11111

d u 1110 00 101 11110 100 011 00 101 010 00 11111 1100 100 1101

Slide credit: Ashish Ghosh

Some Basic Compression Methods – Huffman Coding (Block Code)

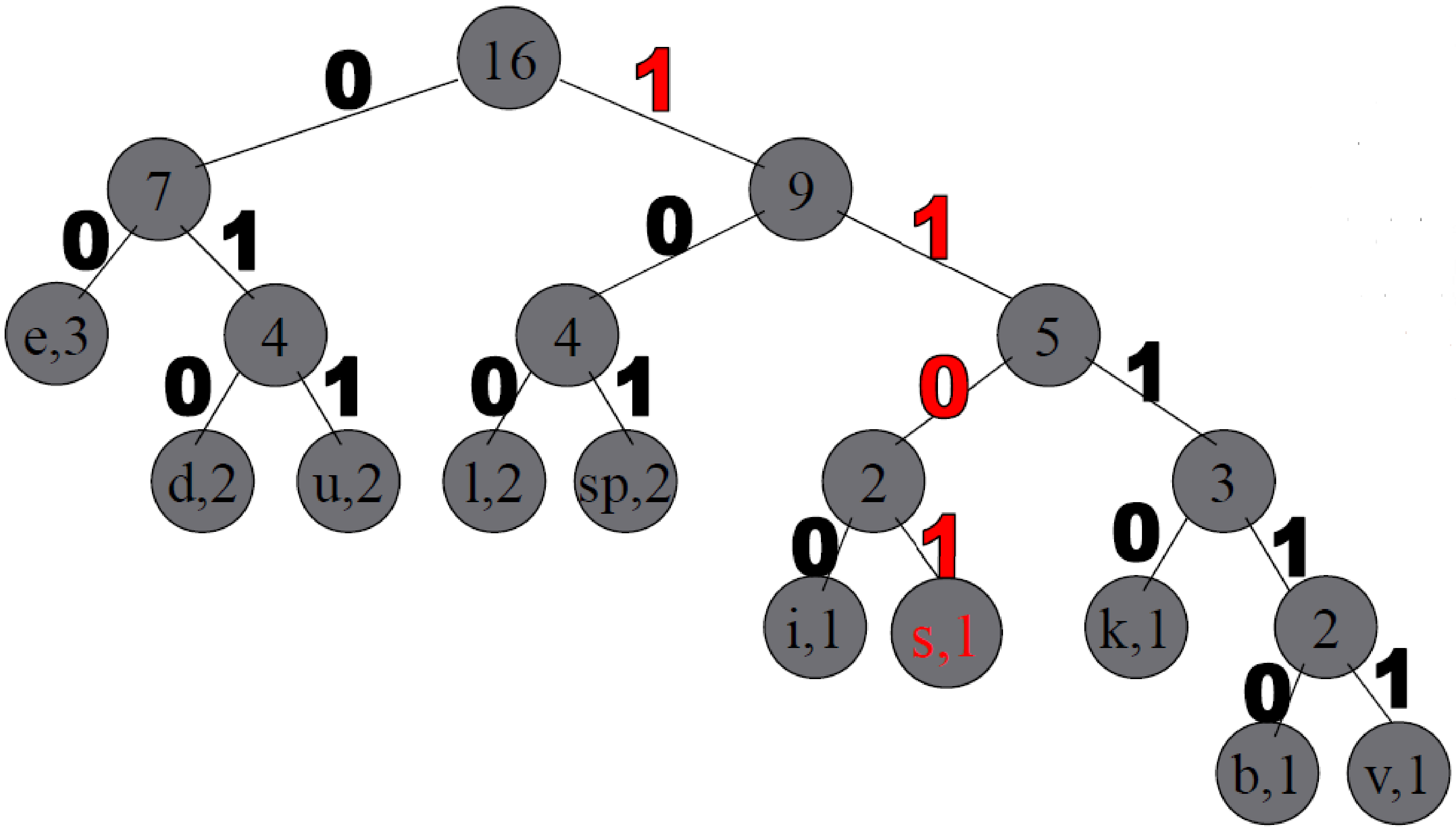


e	00
d	010
u	011
l	100
sp	101
i	1100
s	1101
k	1110
b	11110
v	11111

d u k e b l u e d e v i l 1101

Slide credit: Ashish Ghosh

Some Basic Compression Methods – Huffman Coding (Block Code)

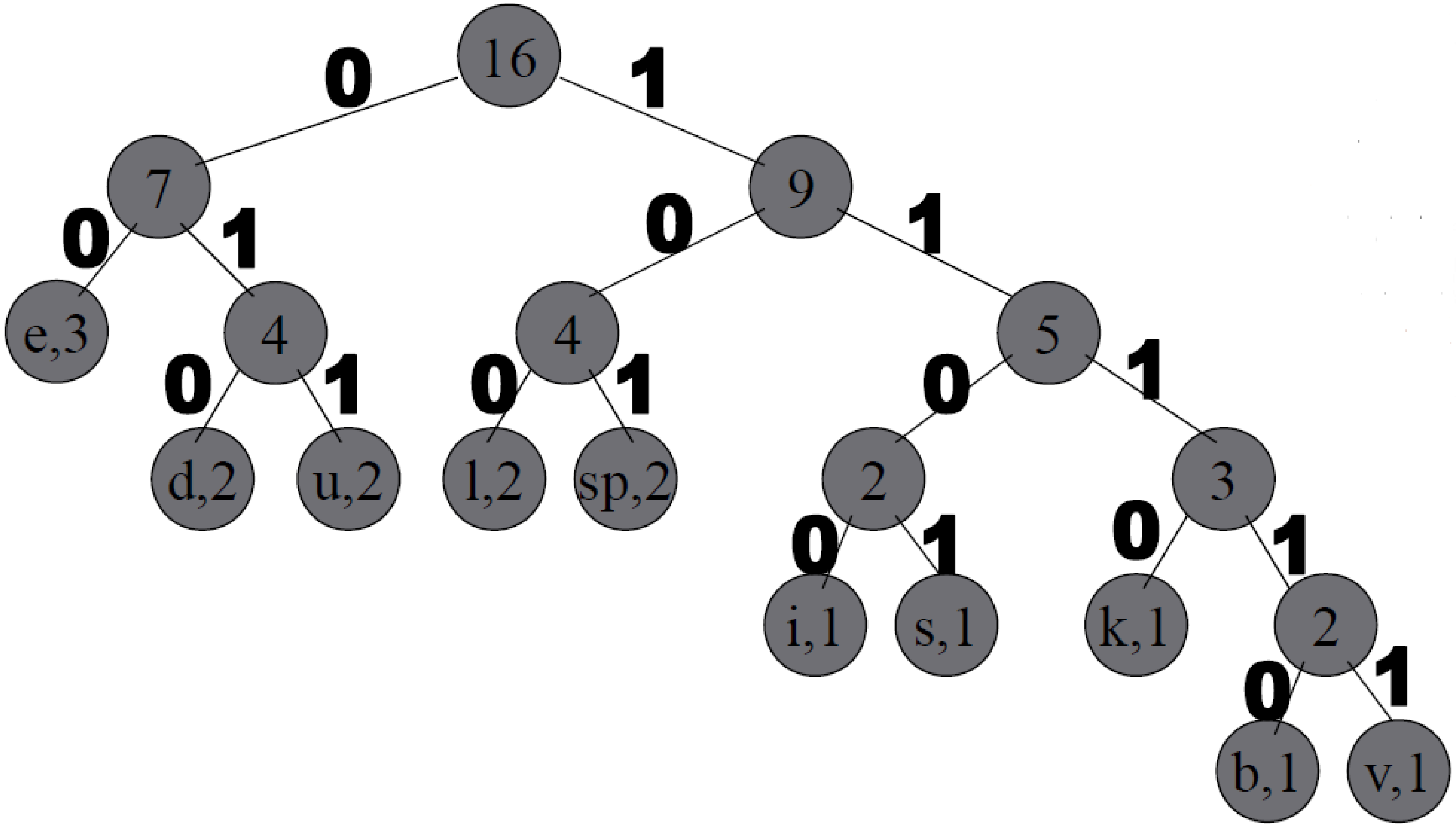


e	00
d	010
u	011
l	100
sp	101
i	1100
s	1101
k	1110
b	11110
v	11111

d u k e b l u e d e v i l s

Slide credit: Ashish Ghosh

Some Basic Compression Methods – Huffman Coding (Block Code)



e	00
d	010
u	011
l	100
sp	101
i	1100
s	1101
k	1110
b	11110
v	11111

010 011 1110 00 101 11110 100 011 00 101 010 00 11111 1100 100 1101

→ d u k e b l u e d e v i l s

Slide credit: Ashish Ghosh

Some Basic Compression Methods – Huffman Coding (Block Code)

- Remove coding redundancy
- Used widely in CCITT group3, JBIG2, JPEG, and MPEG
- Create the optimal code for a set of symbols

Original source		Source reduction			
Symbol	Probability	1	2	3	4
a_2	0.4	0.4	0.4	0.4	0.6
a_6	0.3	0.3	0.3	0.3	0.4
a_1	0.1	0.1	0.2	0.3	
a_4	0.1	0.1	0.1		
a_3	0.06	0.1			
a_5	0.04				

FIGURE 8.7 Huffman source reductions.

Slide credit: Yan Tong

Some Basic Compression Methods – Huffman Coding (Block Code)

Original source			Source reduction							
Symbol	Probability	Code	1		2		3		4	
a_2	0.4	1	0.4	1	0.4	1	0.4	1	0.6	0
a_6	0.3	00	0.3	00	0.3	00	0.3	00	0.4	1
a_1	0.1	011	0.1	011	0.2	010	0.3	01		
a_4	0.1	0100	0.1	0100	0.1	011				
a_3	0.06	01010	0.1	0101						
a_5	0.04	01011								

FIGURE 8.8 Huffman code assignment procedure.

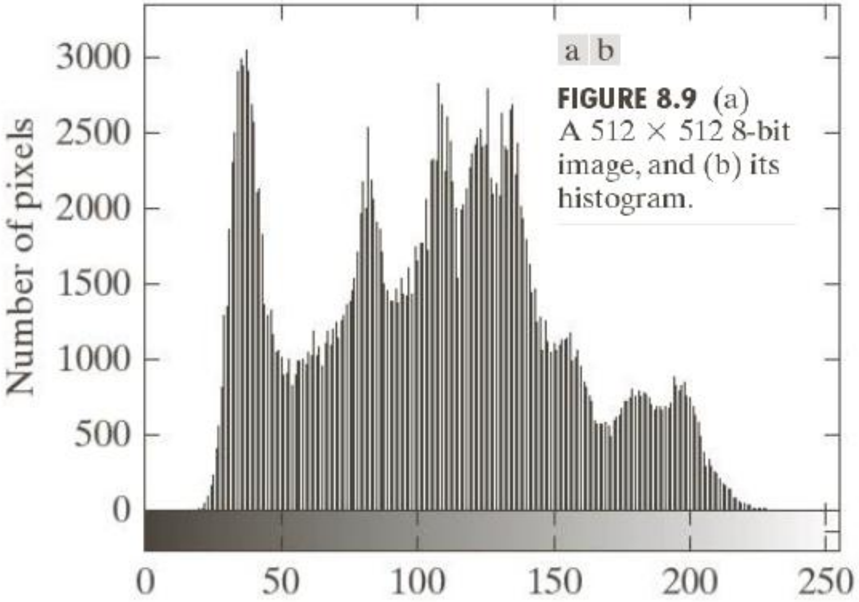
$$L_{\text{avg}} = (0.4)(1) + (0.3)(2) + (0.1)(3) + (0.1)(4) + (0.06)(5) + (0.04)(5) = 2.2\text{bits/pixel}$$

$$H = 2.14\text{bits/pixel}$$

- **Block code**: each source symbol is represented by a fixed code symbol
- **Instantaneous**: lookup table
- **Uniquely decodable**: extract symbols in a left-to-right manner

Slide credit: Yan Tong

Some Basic Compression Methods – Huffman Coding (Block Code)



a b

FIGURE 8.9 (a) A 512 × 512 8-bit image, and (b) its histogram.

→ 7.428 bits/pixel

In practice, a pre-computed Huffman coding table is used (e.g., JPEG and MPEG)

Slide credit: Yan Tong

Arithmetic Coding -- Nonblock Code

- Used in JBIG, JBIG2, JPEG2000, and MPEG4
- Non-block: the whole message is encoded into a single code word (real value in [0, 1])

Source Symbol	Probability	Initial Subinterval
a_1	0.2	[0.0, 0.2)
a_2	0.2	[0.2, 0.4)
a_3	0.4	[0.4, 0.8)
a_4	0.2	[0.8, 1.0)

TABLE 8.6 Arithmetic coding example.

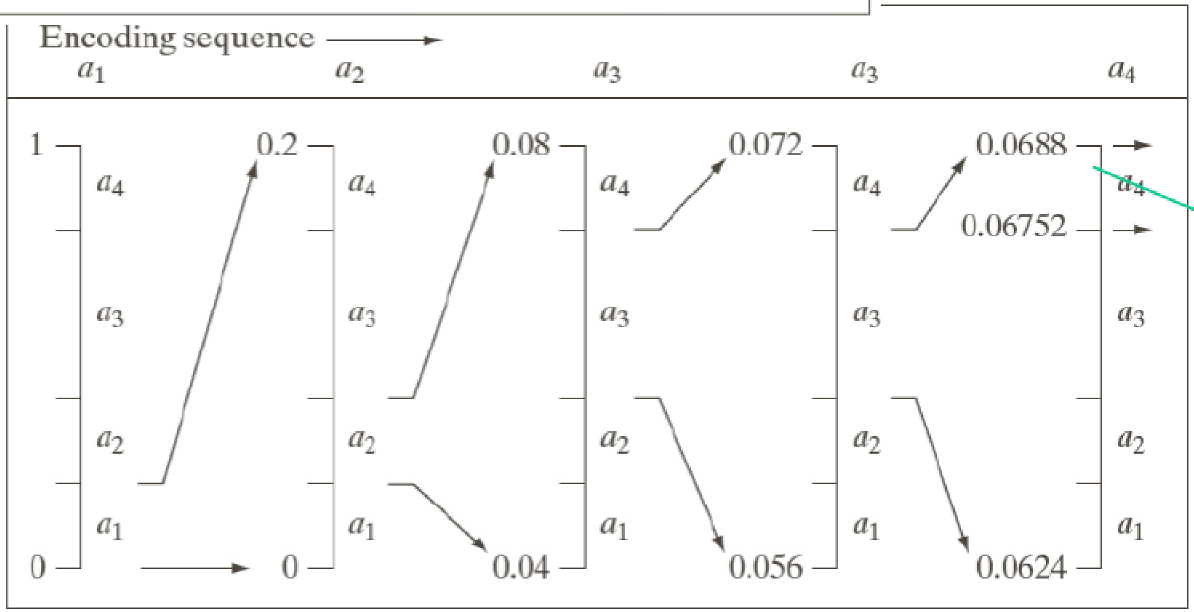


FIGURE 8.12 Arithmetic coding procedure.

Any number in the range can be used

Assume 0.068

Slide credit: Yan Tong

Arithmetic Coding -- Nonblock Code

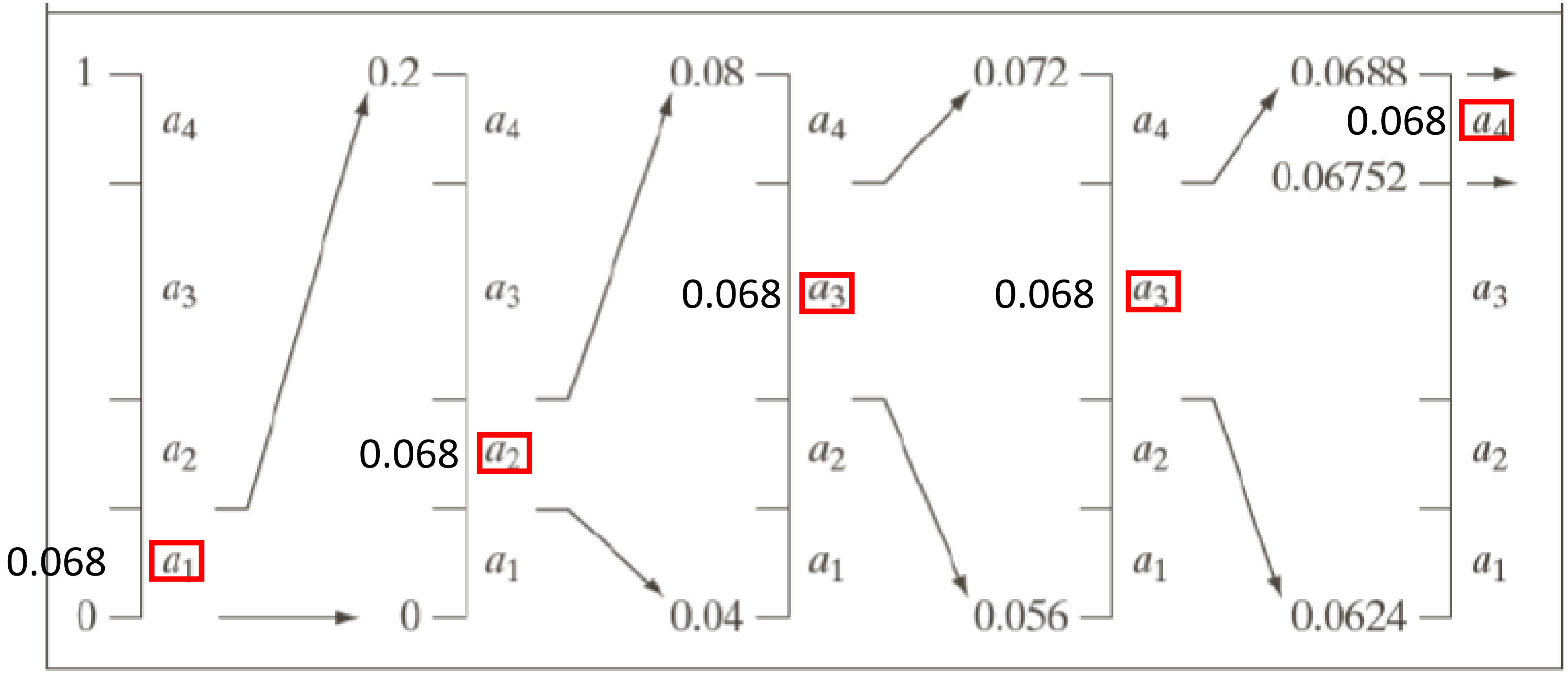
Decoding

- final value
- probabilities of the input symbols

Source Symbol	Probability	Initial Subinterval
a_1	0.2	[0.0, 0.2)
a_2	0.2	[0.2, 0.4)
a_3	0.4	[0.4, 0.8)
a_4	0.2	[0.8, 1.0)

Two decoding methods:

- Straightforward decoding



Slide credit: Yan Tong

Arithmetic Coding -- Nonblock Code

Decoding

- final value
- probabilities of the input symbols

Two decoding methods:

- Straightforward decoding
- An efficient method

Source Symbol	Probability	Initial Subinterval
a_1	0.2	[0.0, 0.2)
a_2	0.2	[0.2, 0.4)
a_3	0.4	[0.4, 0.8)
a_4	0.2	[0.8, 1.0)

Step0: $v_t = v_0$

Repeat:

step1: find symbol s_t satisfying $low(s_t) \le v_t \le up(s_t)$

step2: $$v_{t+1} = \frac{v_t - low(s_t)}{p(s_t)}$$

Until: s_t is the end symbol

Slide credit: Yan Tong

Arithmetic Coding -- Nonblock Code

Require an end-of-message indicator

Potential issues:

- Decoding starts when all the message is received
- Sensitive to the noise during transmission
- Limited by the precision – solved by scaling

Slide credit: Yan Tong

Run Length Coding

- Run-length encoding is probably the simplest method of compression.
- It can be used to compress data made of any combination of symbols.
- It does not need to know the frequency of occurrence of symbols and can be very efficient if data is represented as 0s and 1s.
- **The general idea behind this method is to replace consecutive repeating occurrences of a symbol by one occurrence of the symbol followed by the number of occurrences.**
- The method can be even more efficient if the data uses only two symbols (for example 0 and 1) in its bit pattern and one symbol is more frequent than the other.

Slide credit: Ashish Ghosh

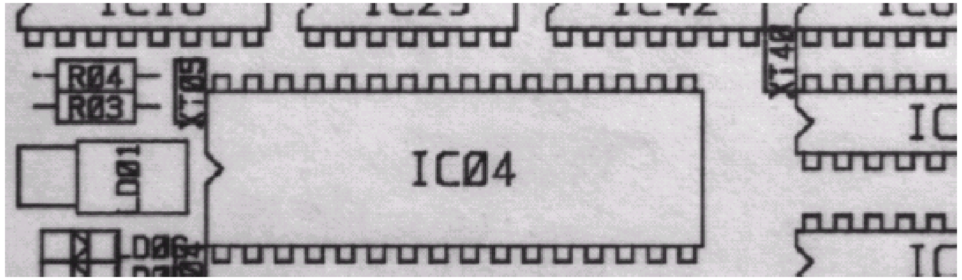
Run Length Coding: Example

- A scan line of a binary image is 00000 00000 00000 00000 00010 00000 00000 01000 00000 00000
- Total of 50 bits
- However, strings of consecutive 0's or 1's can be represented
- More efficiently 0(23) 1(1) 0(12) 1(1) 0(13)
- If the counts can be represented using 5 bits, then we can reduce the amount of data to 5+5*5=30 bits. A compression ratio of 40%

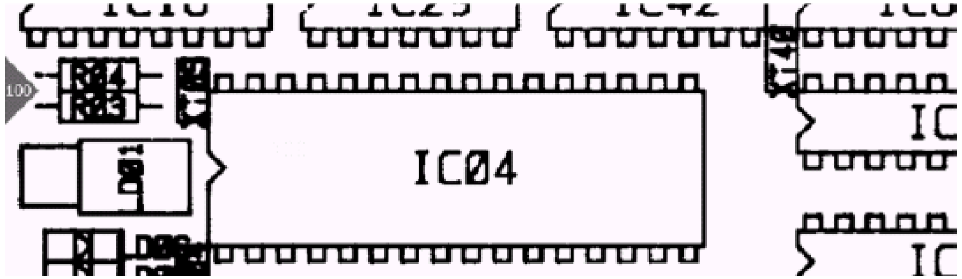
Slide credit: Ashish Ghosh

Run Length Coding: Example

Original

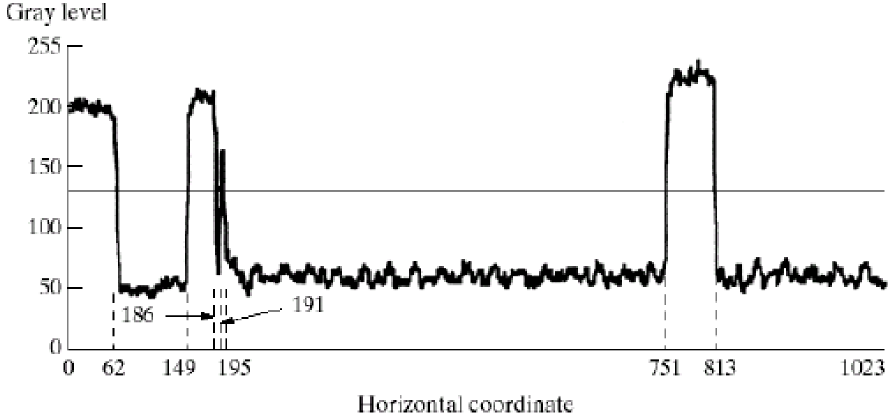


Binary



a
b
c
d

FIGURE 8.3 Illustration of run-length coding: (a) original image. (b) Binary image with line 100 marked. (c) Line profile and binarization threshold. (d) Run-length code.



Run-length

Line 100: (1, 63) (0, 87) (1, 37) (0, 5) (1, 4) (0, 556) (1, 62) (0, 210)

Slide credit: Ashish Ghosh