西安交通大学
XI'AN JIAOTONG UNIVERSITY

第6章 图像压缩

田智强

西安交通大学软件学院

6 图像压缩

6.1 基础知识

6.2 基本压缩方法

一、为什么需要图像压缩?

图像的数据量通常很大，对存储、处理和传输带来许多问题。

✓ **压缩的必要性**

- 1秒标清视频：30帧/秒 × (720×480)像素/帧 × 3字节/像素 = 30MB/秒
- 2小时标清视频：2小时 × (60×60)秒/小时 × 30MB/秒 = 211GB

随着现代通信技术的发展，要求传输的图像信息的种类和数据量越来越大，若不对此进行数据压缩，便难以推广应用。

二、如何进行图像压缩?

✓ 压缩的可能性

1. 数据冗余

数据是信息传递的手段，不同数量的数据可以表示相同数量的信息，包含不相关或重复信息的表示被认为是包含了**冗余数据**，所以减少表示给定信息所需数据量的处理就是**数据压缩**。

二、如何进行图像压缩?

✓ 压缩的可能性

2. 用户通常允许图像失真

用户对原始图像的信号不全都感兴趣，可用特征提取和图像识别的方法，丢掉大量无用的信息，提取有用的信息，使必须传输和存储的图像数据大大减少。

二、如何进行图像压缩?

✓ 图像压缩的实现手段

在数字图像压缩中，可以确定三种基本的数据冗余并加以利用：

1. **编码冗余（Coding Redundancy）**；
2. **空间和时间冗余（Spatial and Temporal Redundancy）**；
3. **不相关信息（Irrelevant Information）**。

如果能减少或消除这三种冗余的一种或多种，就能取得图像压缩的效果。

三、图像压缩技术的重要指标

✓ 压缩率和相对数据冗余

令 b 和 b' 表示相同信息的两种表示中的比特数，则用 b 比特表示的相对数据冗余 $R = 1 - 1/C$。其中，C 称作压缩率，定义为 $C = b/b'$。

例如，$b = 10$，$b' = 1$，压缩率 $C = 10$，相对数据冗余 $R = 1 - 1/10 = 0.9$，表示用 b 表示时，有 90% 的数据都是冗余的。

四、图像压缩的目的

图像数据压缩的目的是，在满足一定图像质量的条件下，用尽可能少的比特数来表示原始图像，以提高图像传输的效率和减少图像存储的容量。

6.1 基础知识

6.1.1 编码冗余

6.1.2 空间冗余和时间冗余

6.1.3 不相关信息

6.1.4 图像信息的度量

6.1.5 保真度准则

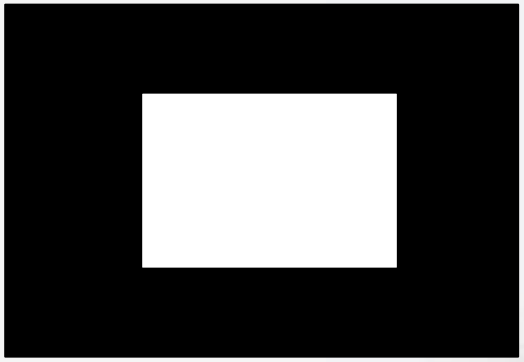
6.1.6 图像压缩模型

6.1.7 图像格式、存储器、压缩标准

6.1.1 编码冗余

1. 什么是编码冗余?

如果一个图像的灰度级编码，使用了多于实际需要的编码符号，就称该图像包含了编码冗余。



比如，用 8 位表示该图像的像素，就称该图像存在编码冗余。因为该图像的像素只有两个灰度，用 1 位即可表示。

多少位表示这个图像的像素值?

6.1.1 编码冗余

假设用区间 $[0, L-1]$ 内的一个离散随机变量 r_k 来表示一幅 $M \times N$ 图像的灰度，且**每个r_k出现的概率为** $p_r(r_k)$，则：

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

其中，L 是灰度级数，n_k 是第 k 级灰度在图像中出现的次数。

6.1.1 编码冗余

若用于表示每个 r_k 值的比特数为 $l(r_k)$，则表示**每个像素所需的平均比特数**为：

$$L_{avg} = \sum_{k=0}^{L-1} l(r_k) p_r(r_k) \qquad (2)$$

那么，表示 $M \times N$ 图像所需的总比特数为 MNL_{avg}

当 $l(r_k)$ 等于常数 m 时，表示使用 自然 m 比特固定长度码 来表示灰度，此时每个 r_k 值的比特数 $l(r_k) = m$，

$L_{avg} =$ m ？

6.1.1 编码冗余

2. 两种编码方式

- **自然 m 比特固定长度编码**：将被编码的每个信息分配来自一个 m 比特二进制计数序列的 2^m 种编码中的一种。

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

- **变长编码**：通过评估字符出现机率，对出现机率高的字符使用较短编码，反之则使用较长编码，从而降低编码后字符串的平均长度，达到压缩的目的。

6.1.1 编码冗余（例）

例：自然m比特固定长度编码和变长编码说明

- Code 1 和 Code 2：8 比特自然二进制编码和变长编码方式
- $p_r(r_k)$：该图的灰度概率分布值
- $l_1(r_k)$ 和 $l_2(r_k)$：两种编码方式需要的比特数

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

6.1.1 编码冗余（例）

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

- **平均比特数**

8 比特自然二进制编码：$L_{avg} = 8$

变长编码？：$L_{avg} = \sum_{k=0}^{L-1} l(r_k)p_r(r_k)$

$$L_{avg} = 0.25 \times 2 + 0.47 \times 1 + 0.25 \times 3 + 0.03 \times 3$$

$$= 1.81$$

6.1.1 编码冗余（例）

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

- **表示整幅图所需比特数**

8 比特自然二进制编码：

$$MNL_{avg} = 256 \times 256 \times 8 = 524288$$

变长编码？ ：

$$MNL_{avg} = 256 \times 256 \times 1.81 = 118621$$

6.1.1 编码冗余（例）

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

- **压缩率C和相对数据冗余R**

$$C = \frac{256 \times 256 \times 8}{256 \times 256 \times 1.81} = \frac{8}{1.81} \approx 4.42$$

$$R = 1 - \frac{1}{4.42} = 0.774$$

77.4%的相对数据冗余

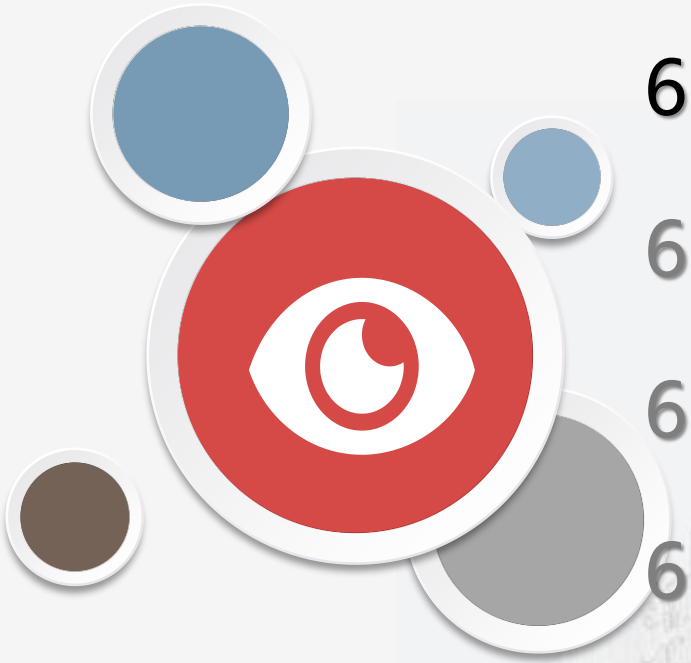
6.1.1 编码冗余

3. 编码冗余产生的原因

- 大多数图像的直方图是不均匀的，这就意味着某些灰度比其他灰度更可能出现；
- 自然二进制编码对最大和最小可能值分配相同的比特数，从而产生了冗余。

用变长编码达到的压缩效果来自较大可能的灰度值分配较少的比特，反之分配较多的比特。

6.1 基础知识



6.1.1 编码冗余

6.1.2 空间冗余和时间冗余

6.1.3 不相关信息

6.1.4 图像信息的度量

6.1.5 保真度准则

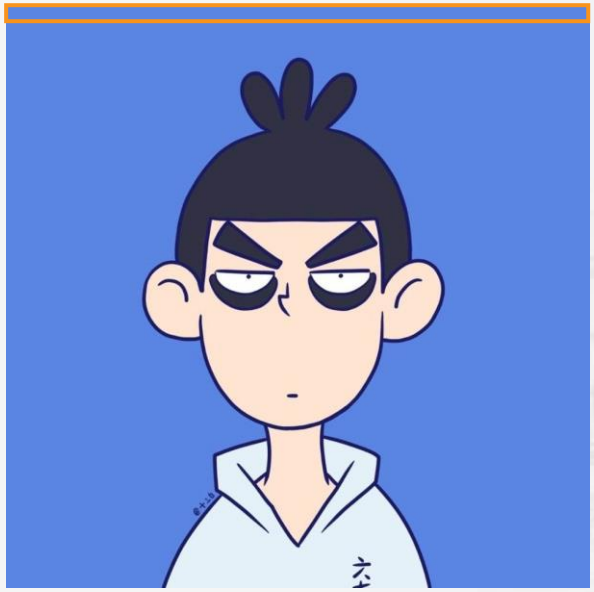
6.1.6 图像压缩模型

6.1.7 图像格式、存储器、压缩标准

6.1.2 空间冗余和时间冗余

1. 空间冗余

一幅图像表面上各采样点的颜色之间往往存在着空间连贯性，这些颜色相同的块就可以压缩。空间冗余主要发生在单张图片中，所以空间冗余也称图像内相关（intra-picture correlation）。



⇦ 第一行256像素点

比如，第一行像素值相同，假设原本存储为[100, 100, ..., 100]，共 256 个像素，则需要空间 1 byte * 256。

为了节省不必要的存储空间，引入行程长度编码。

6.1.2 空间冗余和时间冗余

2. 对空间冗余的处理

行程长度编码(run length encoding, RLE):

一幅图像中有许多颜色相同的图块，此时不需要存储每一个像素的颜色值，仅存储一个像素的颜色值以及具有相同颜色的像素数目即可。具有相同颜色且是连续的像素数目称为行程长度。

比如字符串 "AAAABBBCCDEEEE"，由 4 个 A、3 个 B、2 个 C、1 个 D、4 个 E 组成，通过行程长度编码可压缩为 4A3B2C1D4E。

6.1.2 空间冗余和时间冗余

2.对空间冗余的处理

第一行256个像素点

原表示： [100, 100, ..., 100]，1 byte * 256

行程长度表示：[256, 100]，2 byte

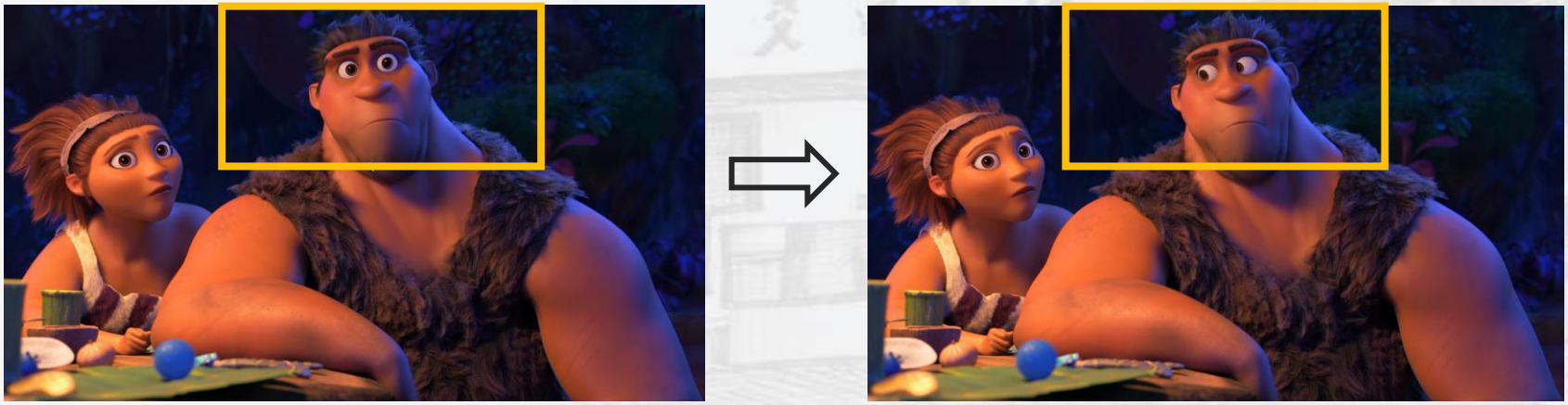
压缩比？：

256 : 2

6.1.2 空间冗余和时间冗余

3. 时间冗余

时间冗余取决于视频中不同帧之间的相关性。为了人类能够感知到平滑、连续的运动，视频经常以超过 24fps 的帧速率显示，这也要求相邻帧之间需要具备一定的相似性。时间冗余是序列图像中经常包含的冗余，所以时间冗余也称图像间相关（inter-picture correlation）。



6.1.2 空间冗余和时间冗余

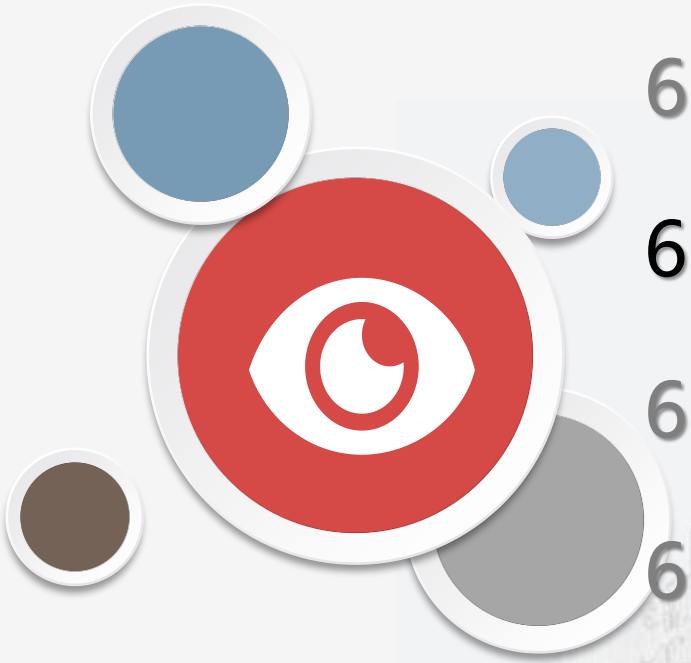
4. 消除时间冗余

利用差异信息重建帧：

由于时间冗余，帧之间的差异信号通常非常小。将差异信号发送到接收端，接收器则可以利用差异信号和已经接收到的参考帧重建视频帧。

重建帧　　参考帧　　差异信息

6.1 基础知识



6.1.1 编码冗余

6.1.2 空间冗余和时间冗余

6.1.3 不相关信息

6.1.4 图像信息的度量

6.1.5 保真度准则

6.1.6 图像压缩模型

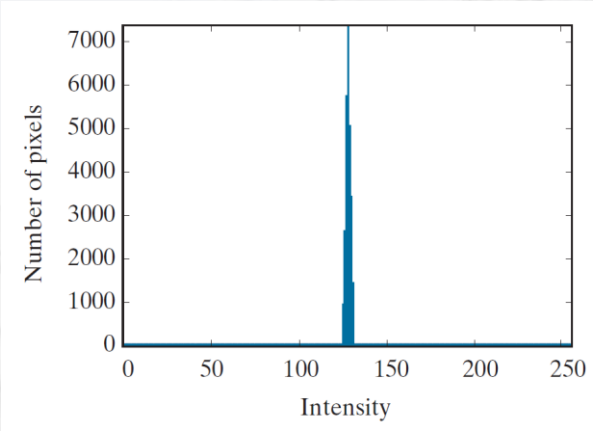
6.1.7 图像格式、存储器、压缩标准

6.1.3 不相关信息

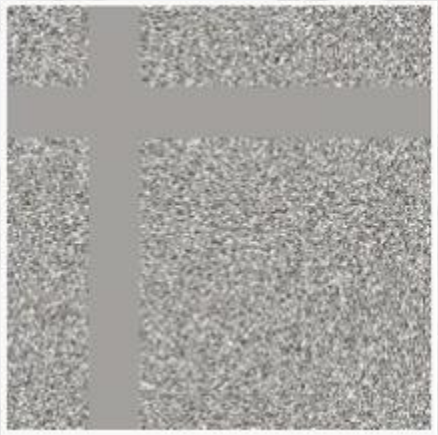
1. 不相关信息

人类视觉系统对图像场的敏感性是非均匀和非线性的。然而，在记录原始的图像数据时，通常假定视觉系统是线性的和均匀的，视觉敏感和不敏感的部分同等对待，从而产生了比理想编码更多的数据，所以不相关信息也称作 视觉冗余。

原图



原图的直方图

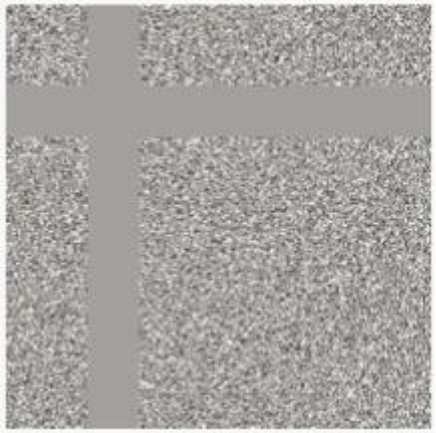


原图经直方图均衡处理后

6.1.3 不相关信息

1. 不相关信息

原图



原图经直方图均衡处理后

被人类视觉系统忽略的信息或与图像预期应用无关的信息都是可被删除的对象。这样原图可由其平均灰度来表示，原本的 $256 \times 256 \times 8$ 比特灰度阵列被压缩到单个字节，得到压缩率 $256 \times 256 \times 8/8 = 65536:1$

6.1.3 不相关信息

2. 对不相关信息的处理

通过 **量化** 操作能够去除无关信息。它意味着将较宽范围的输入值映射为有限数量的输出值，信息因此发生损失。

6.1.3 不相关信息

2. 对不相关信息的处理

量化是一种 **不可逆** 的操作，量化的结果是数据的有损压缩。信息是否保留应由应用决定，如果该信息很重要，如它可能应用于医学中，就不应该遗漏；否则，这种信息就是冗余的。

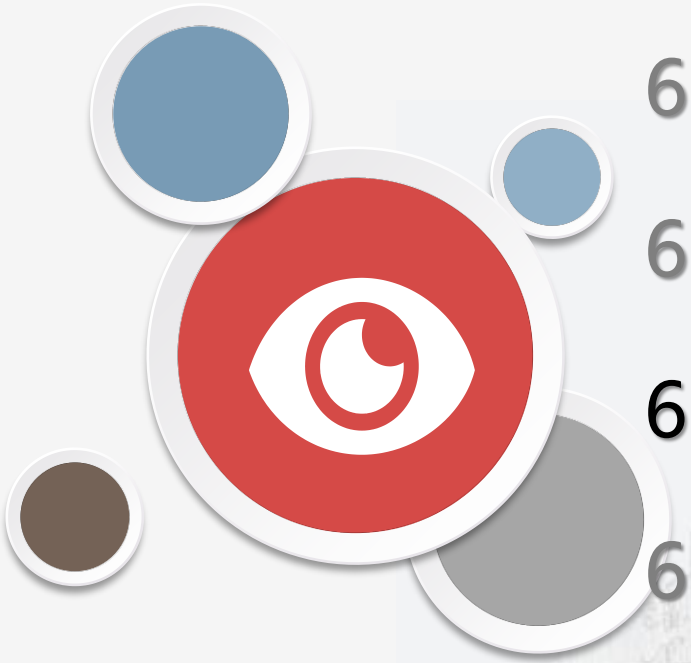
6.1.3 不相关信息

3. 数据冗余对比

视觉冗余本质上是一种主观的数据冗余，而编码冗余和空间、时间冗余是客观存在的，可以通过数学表达式定量地去衡量。

- 编码冗余：不同像素值出现的概率不同；
- 空间冗余：图像相邻像素之间有较强的相关性；
- 时间冗余：视频序列的相邻图像之间内容相似；
- 不相关信息（视觉冗余）：人的视觉系统对某些细节不敏感。

6.1 基础知识



6.1.1 编码冗余

6.1.2 空间冗余和时间冗余

6.1.3 不相关信息

6.1.4 图像信息的度量

6.1.5 保真度准则

6.1.6 图像压缩模型

6.1.7 图像格式、存储器、压缩标准

6.1.4 图像信息的度量

图像压缩中的问题：

- 是否存在可充分描述一幅图像的最小数据量？

信息论——回答这类问题的理论框架

信息论的基本前提是，信息的产生可用一个概率过程建模。那么，对于随机事件，它的出现会给人们带来多大的信息量？

6.1.4 图像信息的度量

1. 信息量

随机事件 E 的概率 $P(E)$ 包含 $I(E)$ 单位的信息，其中：

$$I(E) = log\frac{1}{P(E)} = -logP(E) \qquad (3)$$

如果 $P(E) = 1$（即事件总会发生），则 $I(E) = 0$，即认为事件 E 没有信息，因为没有与该事件相关的不确定性，所以在通信中不会传递关于这个事件已经发生的任何信息。

信息量 $I(E)$ 实际上是无量纲的，为了研究问题的方便，**通常根据对数的底定义信息量的量纲**。如果使用以 m 为底的对数，则这种度量称为 m 元单位。

6.1.4 图像信息的度量

1. 信息量

随机事件 E 的概率 $P(E)$ 包含 $I(E)$ 单位的信息，其中：

$$I(E) = log\frac{1}{P(E)} = -logP(E) \qquad (3)$$

- 对数的底取 2，则信息量的单位为比特（bit）；
- 取 e（自然对数），则单位为奈特（nat）；
- 取 10（常用对数），则单位为哈特（hart）。

6.1.4 图像信息的度量（例）

1. 信息量

例：一个 0，1 等概率的二进制随机序列，求底为2的自信息量（单位：比特）。

解：等概率取值为 0 或 1，$P(0)=P(1)=\frac{1}{2}$，所以，

$$I(0)=I(1)=-log\frac{1}{2}=1 \text{ bit}$$

事件的信息量只与其概率有关，而与它的取值无关。

6.1.4 图像信息的度量

2. 信源熵

从一个可能事件的离散集合 $\{a_1, a_2, \ldots, a_J\}$，给定一个统计独立随机事件的信源，与该集合相联系的概率为 $\{P(a_1), P(a_2), \ldots, P(a_J)\}$，则每个信源输出的平均信息称为该信源的熵，即：

$$H = -\sum_{j=1}^{J} P(a_j) log P(a_j) \qquad (4)$$

其中，a_j 称为**信源符号**。由于他们是统计独立的，所以信源本身称为**零记忆信源**。

6.1.4 图像信息的度量（例）

2. 信源熵

将图像考虑为一个虚构的零记忆“灰度信源”的输出，通过观察图像的直方图估计该信源的符号a_j概率，此时，灰度信源的熵为：

$$\tilde{H} = -\sum_{k=0}^{L-1} P_r(r_k) log_2 P_r(r_k) \qquad (5)$$

以 2 为底数，式（5）是以比特度量的虚构灰度信源的每个灰度输出的平均信息。用少于 $\tilde{H}$ 比特/像素的熵对虚构信源的灰度值进行编码是不可行的。

6.1.4 图像信息的度量（例）

2. 信源熵

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

例：图像的熵的估计

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

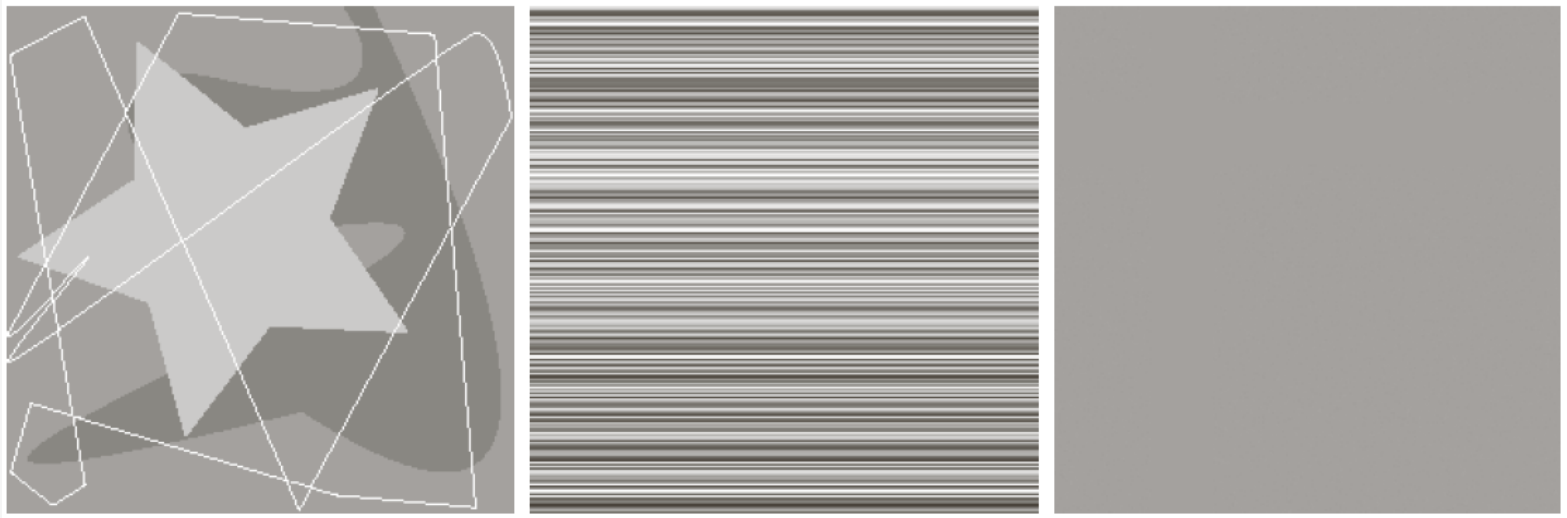
通过将表中的灰度概率 $p_r(r_k)$ 带入式（5）中估计图像的熵：

$$\tilde{H} = -[0.25log_2 0.25 + 0.47log_2 0.47 + 0.25log_2 0.25 + 0.03log_2 0.03]$$

$$\approx 1.6614 \text{ 比特/像素}$$

6.1.4 图像信息的度量（例）

2. 信源熵



(a)　　(b)　　(c)

H = 1.66 bit/pixel　　H = 8 bit/pixel　　H = 1.56 bit/pixel

一幅图像中熵的数量和信息与直觉相差甚远。

6.1.4 图像信息的度量

3. 香农第一定理

香农第一定理（无噪声编码定理）：

$$\lim_{n\to\infty}[\frac{L_{avg,n}}{n}] = H \qquad (6)$$

其中，$L_{avg,n}$ 是表示所有 n 个符号组所需编码符号的平均数。

香农第一定理给出了**在无损条件下**，数据压缩的**临界值**。不论采用何种格式无损保存图片，所需的比特数都大于香农第一定理所给出的值。若低于该临界值，则不可能做到无损压缩。

6.1 基础知识

6.1.1 编码冗余

6.1.2 空间冗余和时间冗余

6.1.3 不相关信息

6.1.4 图像信息的度量

6.1.5 保真度准则

6.1.6 图像压缩模型

6.1.7 图像格式、存储器、压缩标准

6.1.5 保真度准则

去除不相关信息时，涉及定量的图像信息的损失，因此需要对这种丢失进行量化，以描述解码图像相对于原始图像的偏离程度，这些测度称为保真度准则。

常用保真度准则分为两大类：

- **客观保真度准则**——定量描述
- **主观保真度准则**——定性或定性基础上的定量描述

6.1.5 保真度准则

1. 客观保真度准则

如果信息损失的程度可以表示为原始或输入图像与压缩后又解压缩输出的图像的函数，这个函数就被称为客观保真度准则。

令 $f(x,y)$ 为输入图像， $\hat{f}(x,y)$ 为 $f(x,y)$ 的近似，它是对输入图像先压缩后解压缩的结果。对于任意的 x 和 y 值， $f(x,y)$ 和 $\hat{f}(x,y)$ 之间的误差 $\mathrm{e}(x,y)$ 为：

$$e(x,y)=\hat{f}(x,y)-f(x,y) \qquad (7)$$

6.1.5 保真度准则

1. 客观保真度准则

$M \times N$ 大小的两幅图像间的总误差为：

$$\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}[\hat{f}(x,y)-f(x,y)] \qquad (8)$$

常用的两种客观保真度准则：

- 均方根误差
- 均方信噪比

6.1.5 保真度准则

1. 客观保真度准则

$\hat{f}(x,y)$ 和 $f(x,y)$ 之间的均方根（rms）误差 e_{rms} 是在 $M \times N$ 阵列上平均均方误差的平方根：

$$e_{rms} = \left[\frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}[\hat{f}(x,y) - f(x,y)]^2\right]^{\frac{1}{2}} \qquad (9)$$

6.1.5 保真度准则

1. 客观保真度准则

输出图像的均方信噪比 SNR_{ms}：

$$SNR_{ms} = \frac{\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\hat{f}(x,y)^2}{\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}[\hat{f}(x,y)-f(x,y)]^2} \quad (10)$$

在实际应用中，解压缩后的图像最终是由人来观察的。因此，使用人的主观评估来衡量图像质量通常更为适当。

6.1.5 保真度准则

1. 客观保真度准则

(a)　　(b)　　(c)

上图显示了对原图的三种不同的近似，计算均方根误差分别为 5.17，15.67，14.17。所以根据客观保真度准则，这三幅图像按质量递减的顺序排列为(a)，(c)，(b)。

6.1.5 保真度准则

1. 客观保真度准则

(a) (b) (c)

但从人的视觉上看，(b) 比 (c) 保留了更多重要的信息。

6.1.5 保真度准则

2. 主观保真度准则

有时，客观保真度完全一样的两幅图像可能会有完全不同的视觉质量，所以又规定了主观保真度准则，这种方法是把图像显示给观察者，然后把评价结果加以平均，以此来评价一幅图像的主观质量。

评估可使用一个**绝对等级尺度**或借助于 $f(x,y)$ 和 $\hat{f}(x,y)$ 的**并排比较**来获得。

6.1.5 保真度准则

2. 主观保真度准则

值	等级	描述
1	优秀	图像质量非常好，如同人希望的一样好
2	良好	图像质量高，观看舒服，有干扰但不影响观看
3	可用	图像质量可接受，有干扰但不太影响观看
4	刚可看	图像质量差，干扰有些妨碍观看，希望改进
5	差	图像质量很差，几乎无法观看
6	不能用	图像质量极差，不能观看

6.1 基础知识

6.1.1 编码冗余

6.1.2 空间冗余和时间冗余

6.1.3 不相关信息

6.1.4 图像信息的度量

6.1.5 保真度准则

6.1.6 图像压缩模型

6.1.7 图像格式、存储器、压缩标准

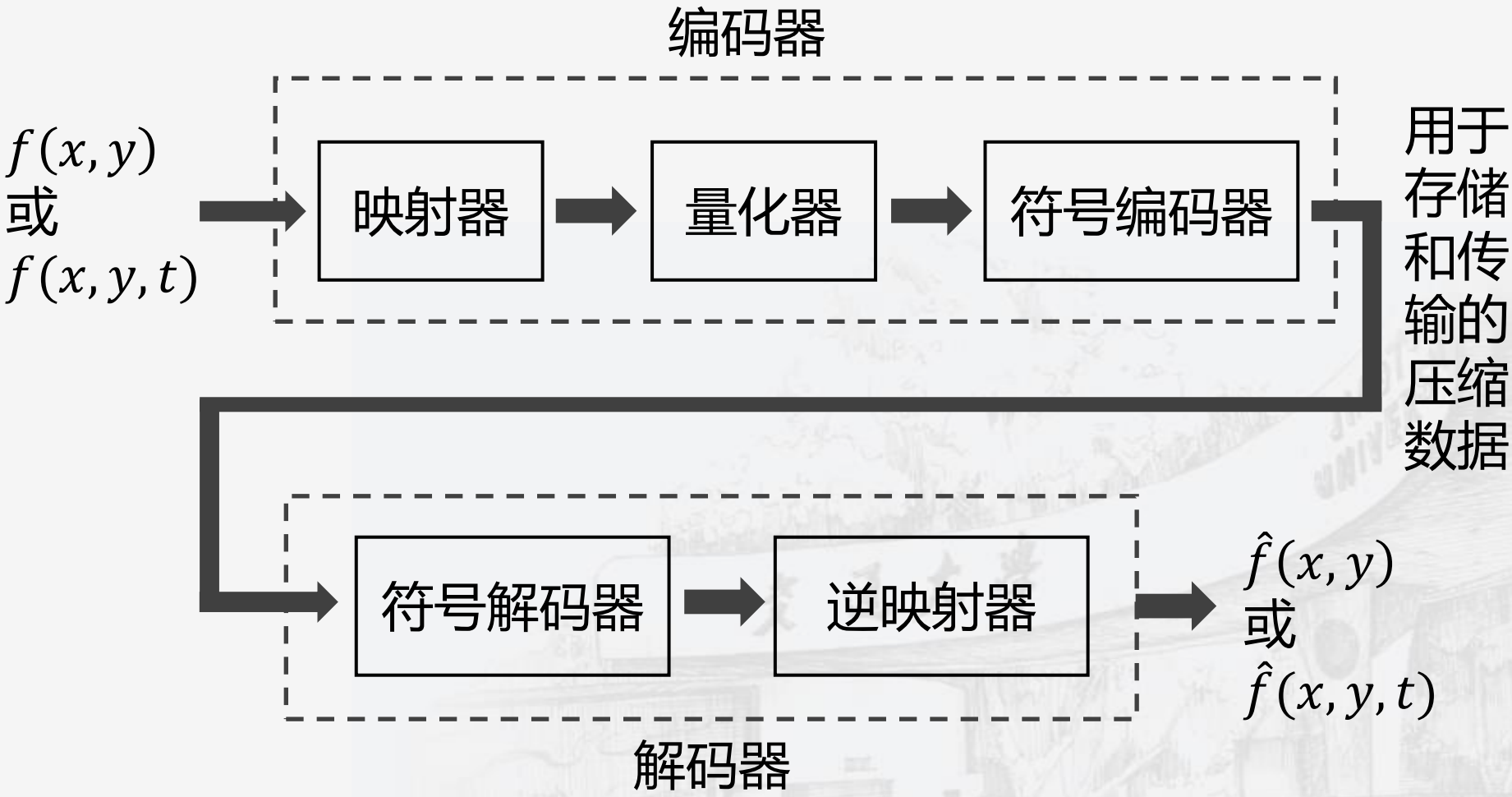
6.1.6 图像压缩模型

图像压缩系统由**编码器**和**解码器**两部分组成。编码器执行压缩操作，解码器执行互补的解压缩操作。

在静止图像应用中，编码器输入和解码器输出分别是 $f(x,y)$ 和 $\hat{f}(x,y)$。在视频应用中，分别是 $f(x,y,t)$ 和 $\hat{f}(x,y,t)$，其中参数 t 为时间。

若 $\hat{f}(x,y)$ 是 $f(x,y)$ 的精确复制品，则称该压缩系统为**无误差的、无损的**或**信息保持的**压缩系统；否则重建的输出图像就会失真，并称该压缩系统为**有损压缩系统**。

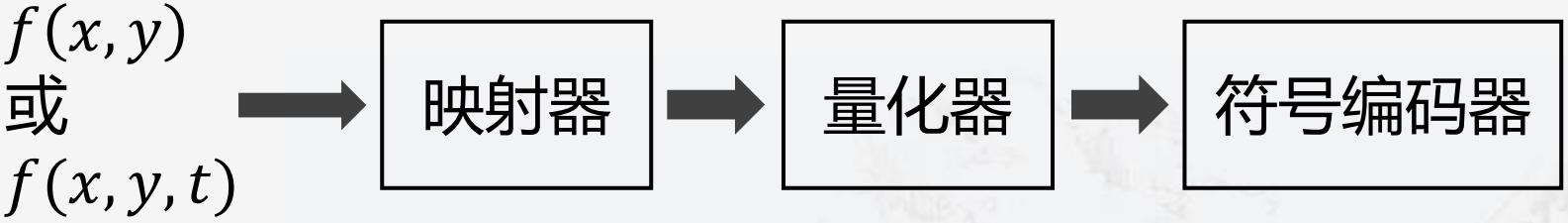
6.1.6 图像压缩模型



通用图像压缩系统功能方框图

6.1.6 图像压缩模型

1. 信源编码器

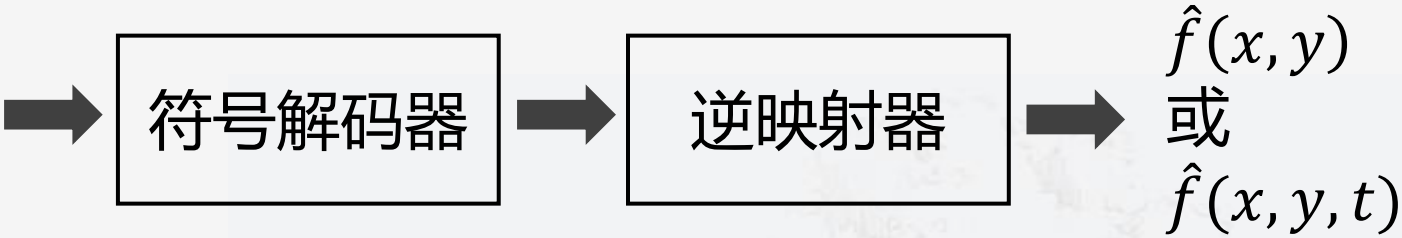


信源编码器：减少或消除输入图像中的编码冗余、时间和空间冗余及不相关信息。

- 映射器：减少时间和空间冗余，该步操作**可逆**
- 量化器：减少不相关信息，该步操作**不可逆**
- 符号编码器：减少编码冗余，该步操作**可逆**

6.1.6 图像压缩模型

2. 信源解码器



信源解码器：以相反的顺序执行编码器的符号编码器和映射器的反操作。

- 符号解码器：进行符号编码的逆操作
- 逆映射器：进行映射器的逆操作

6.1 基础知识

6.1.1 编码冗余

6.1.2 空间冗余和时间冗余

6.1.3 不相关信息

6.1.4 图像信息的度量

6.1.5 保真度准则

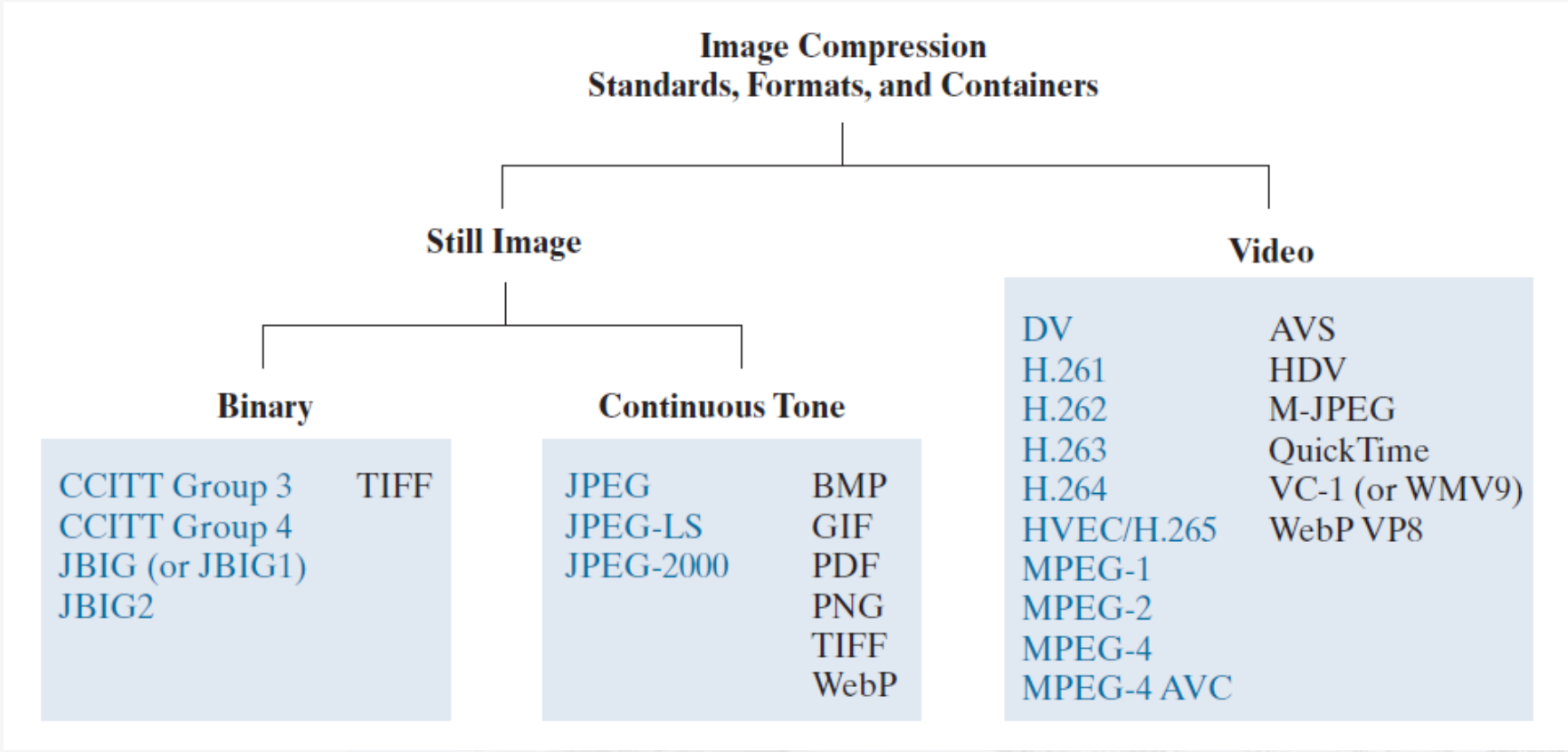
6.1.6 图像压缩模型

6.1.7 图像格式、存储器、压缩标准

6.1.7 图像格式、容器和压缩标准

图像文件格式是组织和存储图像数据的标准方法。它定义了所使用的数据排列的方式和压缩的类型。图像容器类似于文件格式，但处理多种类型的图像数据。图像压缩标准对压缩和解压缩定义了过程。常用的图像格式包括 JPEG，GIF，PNG，TIFF，BMP 等。

6.1.7 图像格式、 容器和压缩标准



更多图像压缩标准、文件格式和容器见数字图像处理（第三版）8.1.7小节

6.2 基本压缩方法

6.2.1 压缩编码方法分类

6.2.2 霍夫曼编码

6.2.3 Golomb编码

6.2.4 算术编码

6.2.5 LZW编码

6.2.1 压缩编码方法分类

图像压缩编码方法分类（根据年代）

- **第一代压缩编码**

以信息论和数字信号处理技术为理论基础，旨在去除图像数据中的**线性相关性**。针对编码冗余和时间空间冗余，压缩比大约在10：1左右。

主要包括**预测编码**、**变换编码**、**统计编码**三种。

6.2.1 压缩编码方法分类

图像压缩编码方法分类（根据年代）

- **第一代压缩编码 - 统计编码**

统计编码主要针对无记忆信源，根据信息码字出现概率的分布特征而进行压缩编码，寻找概率与码字长度间的最优匹配。其又可分为定长码和变长码。

统计编码包括**霍夫曼编码**、**算术编码**、**行程编码**等。

6.2.1 压缩编码方法分类

图像压缩编码方法分类（根据年代）

- **第二代压缩编码**

不局限于信息论的框架，要求充分利用人的视觉生理心理和图像信源的各种特征。其压缩比多在 30 : 1 至 70 : 1 之间。

包括子带编码、基于方向性分解的编码、基于区域分割与合并的编码。

6.2.1 压缩编码方法分类

图像压缩编码方法分类（根据压缩对象）

- **静态图像压缩编码**

包括二值图像、灰度图像、彩色图像的压缩编码。常见的统计编码、变换编码都是针对静态图像的。

- **运动图像压缩编码**

运动图像（视频）除静态图像压缩编码之外，还采用预测编码。

6.2.1 压缩编码方法分类

图像压缩编码方法分类（从信息损失的角度）

- **无损压缩编码**

包括统计编码（霍夫曼编码、算术编码、行程编码、Golomb编码、LZW编码等）、无损预测编码等。

- **有损压缩编码**

包括有损预测编码、变换编码、小波编码等。

6.2 基本压缩方法

6.2.1 压缩编码方法分类

6.2.2 霍夫曼编码

6.2.3 Golomb编码

6.2.4 算数编码

6.2.5 LZW编码

6.2.2 霍夫曼编码

霍夫曼编码是一种利用**信息符号概率分布特性**的可变字长的编码方法。对于出现概率大的信息符号编以短字长的码，对于出现概率小的信息符号编以长字长的码。霍夫曼编码是**消除编码冗余**的最常用技术之一。

6.2.2 霍夫曼编码

霍夫曼编码步骤

1. 将信源符号按出现概率从大到小排成一列，然后把最末两个符号的概率相加，合成一个概率。重复上述做法，直到最后剩下两个概率为止。
2. 从最后一步剩下的两个概率开始逐步向前进行编码。每步只需对两个分支各赋予一个二进制码，如对概率大的赋予码 0，对概率小的赋予码 1（反之亦可）。

6.2.2 霍夫曼编码

霍夫曼编码步骤（例）

1. 将信源符号按出现概率从大到小排成一列。（1）

输入	输入概率
a_2	0.4
a_6	0.3
a_1	0.1
a_4	0.1
a_3	0.06
a_5	0.04

6.2.2 霍夫曼编码

霍夫曼编码步骤（例）

1. 将最末两个符号的概率相加，合成一个概率。（2）

输入	输入概率	第一步
a_2	0.4	0.4
a_6	0.3	0.3
a_1	0.1	0.1
a_4	0.1	0.1
a_3	0.06	0.1
a_5	0.04	

6.2.2 霍夫曼编码

霍夫曼编码步骤（例）

1. 重复上述做法，直到最后剩下两个概率为止。（3）

输入	输入概率	第一步	第二步
a_2	0.4	0.4	0.4
a_6	0.3	0.3	0.3
a_1	0.1	0.1	0.2
a_4	0.1	0.1	0.1
a_3	0.06	0.1	
a_5	0.04		

6.2.2 霍夫曼编码

霍夫曼编码步骤（例）

1. 重复上述做法，直到最后剩下两个概率为止。 (4)

输入	输入概率	第一步	第二步	第三步
a_2	0.4	0.4	0.4	0.4
a_6	0.3	0.3	0.3	0.3
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			

6.2.2 霍夫曼编码

霍夫曼编码步骤（例）

1. 重复上述做法，直到最后剩下两个概率为止。（5）

输入	输入概率	第一步	第二步	第三步	第四步
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				

6.2.2 霍夫曼编码

霍夫曼编码步骤（例）

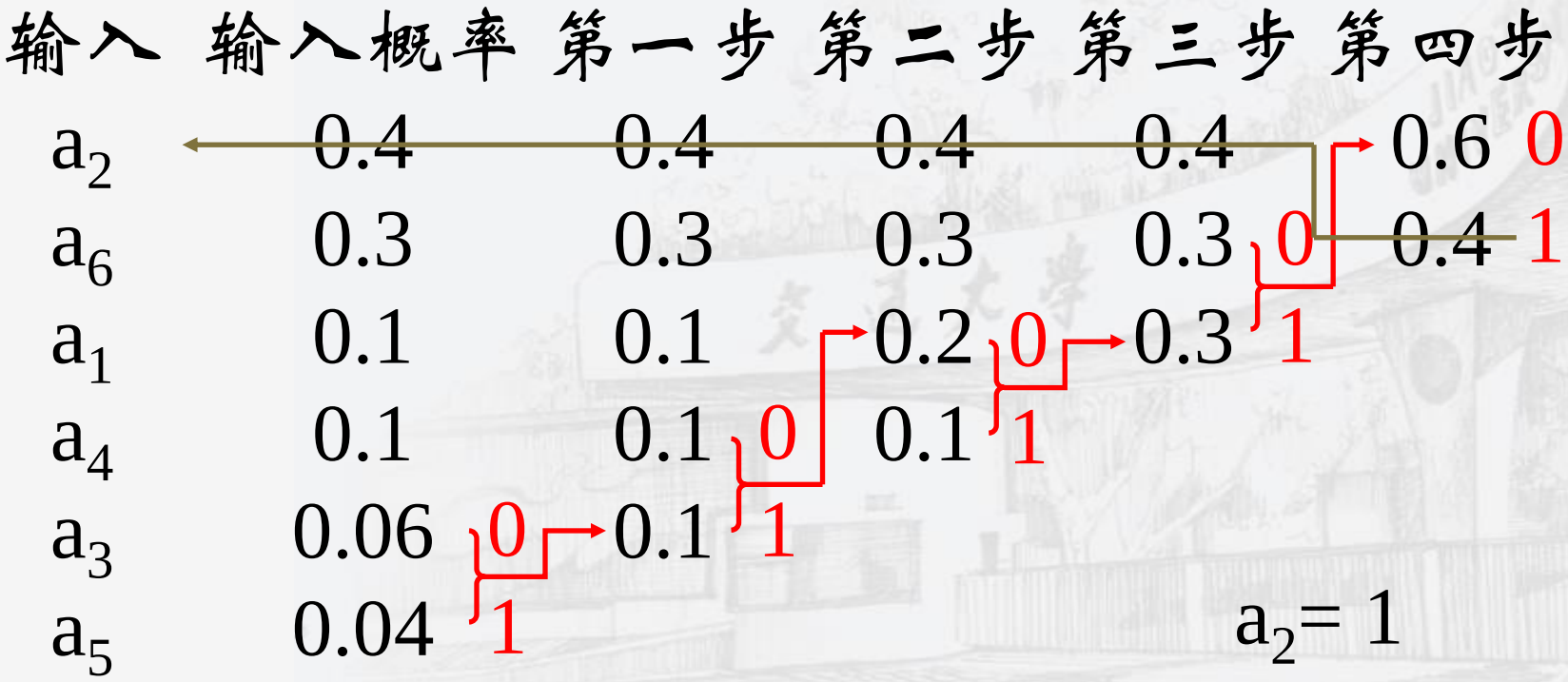
2. 从最后一步剩下的两个概率开始逐步向前进行编码。（6）

输入	输入概率	第一步	第二步	第三步	第四步
a_2	0.4	0.4	0.4	0.4	0.6 0
a_6	0.3	0.3	0.3	0.3 0	0.4 1
a_1	0.1	0.1	0.2 0	0.3 1	
a_4	0.1	0.1 0	0.1 1		
a_3	0.06 0	0.1 1			
a_5	0.04 1				

6.2.2 霍夫曼编码

霍夫曼编码步骤（例）

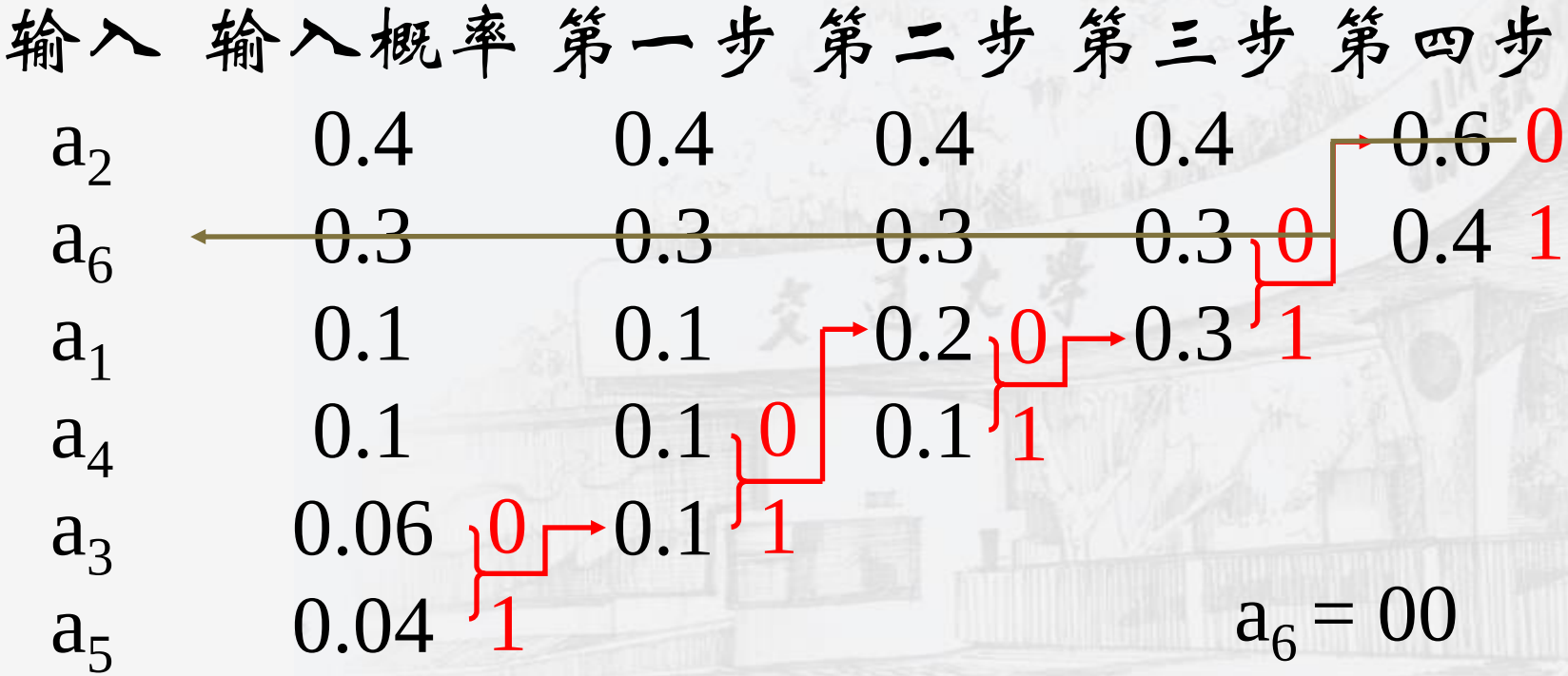
2. 从最后一步剩下的两个概率开始逐步向前进行编码。（7）



6.2.2 霍夫曼编码

霍夫曼编码步骤（例）

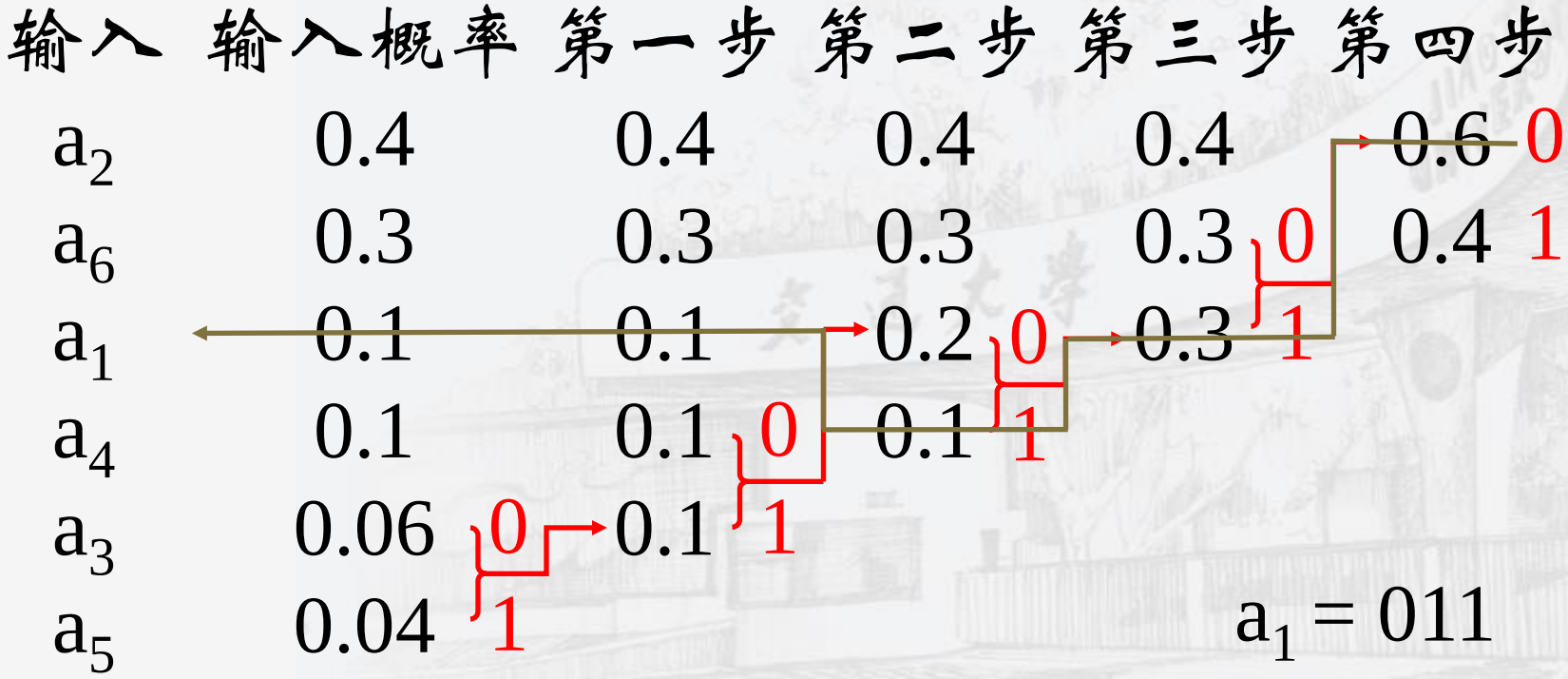
2. 从最后一步剩下的两个概率开始逐步向前进行编码。（8）



6.2.2 霍夫曼编码

霍夫曼编码步骤（例）

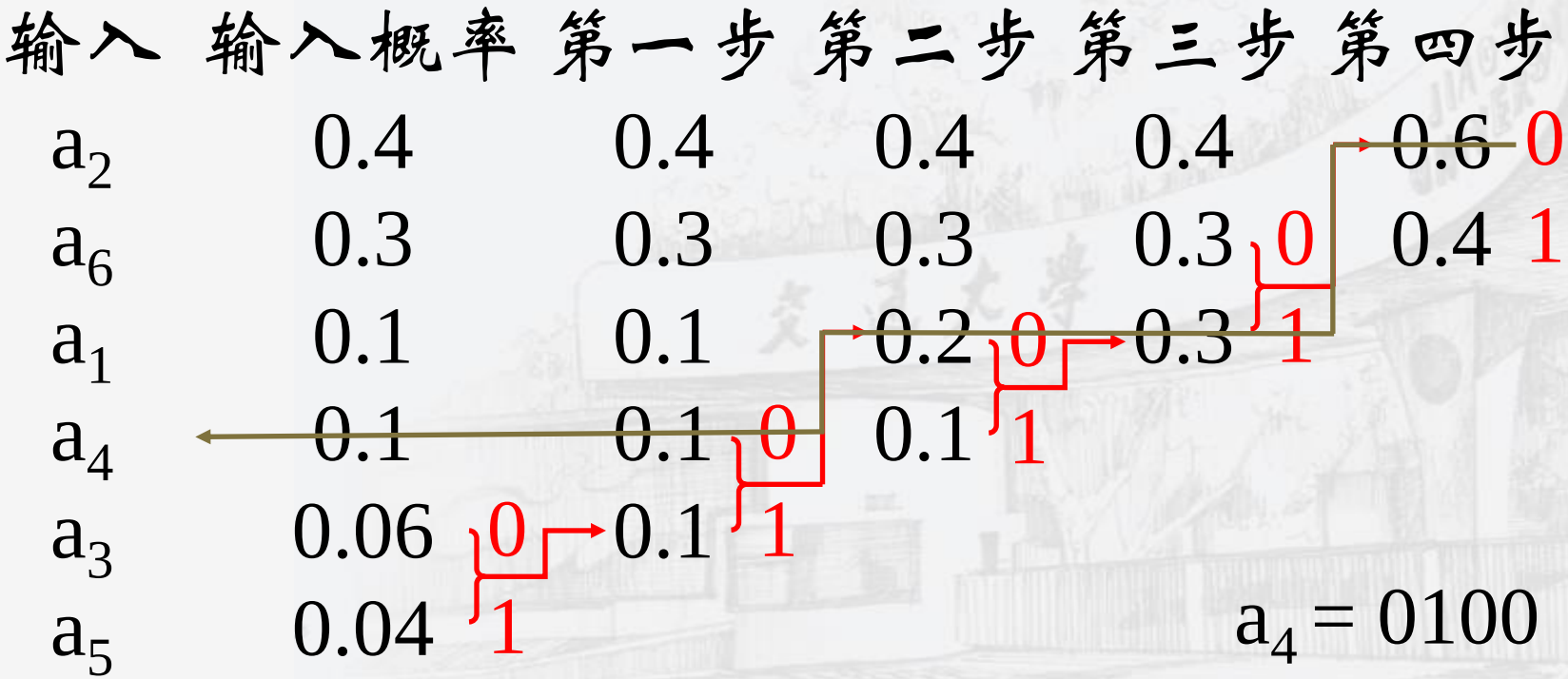
2. 从最后一步剩下的两个概率开始逐步向前进行编码。（9）



6.2.2 霍夫曼编码

霍夫曼编码步骤（例）

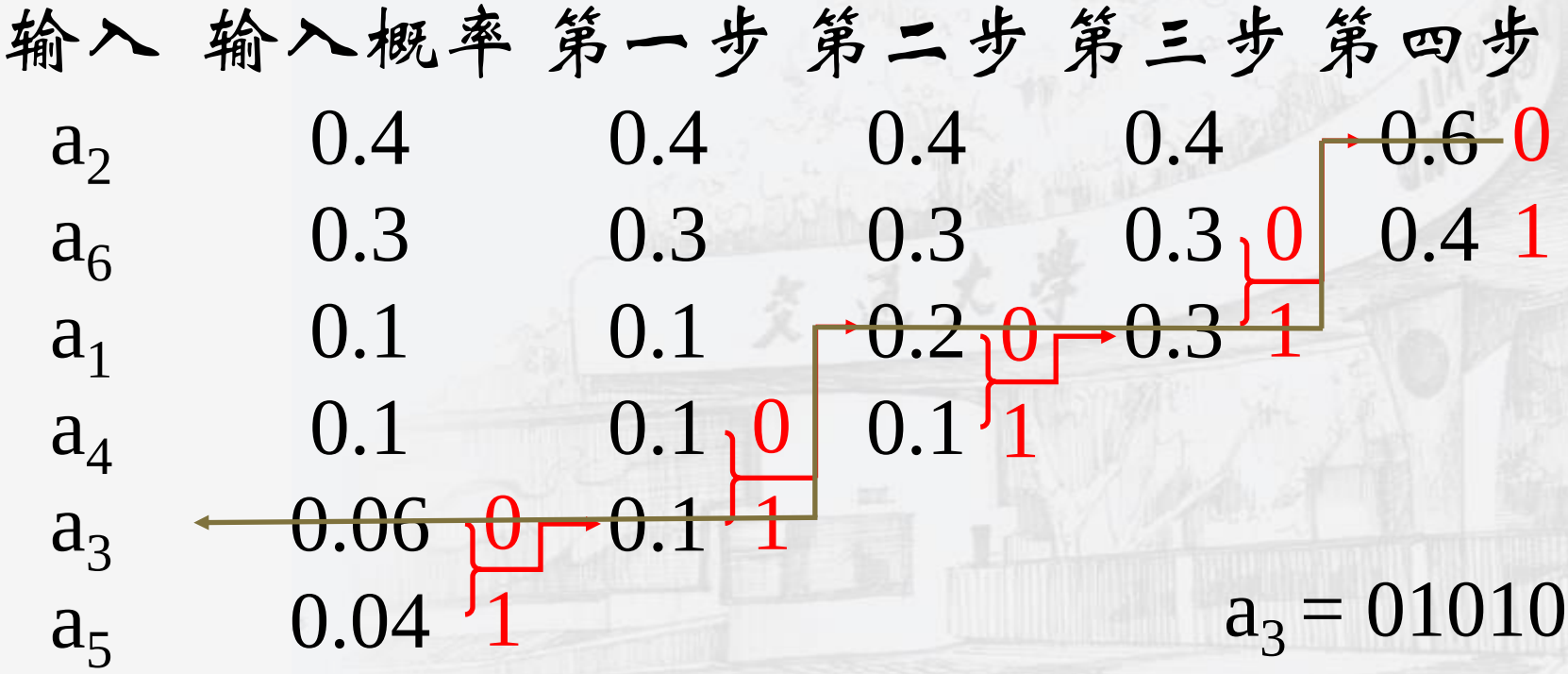
2. 从最后一步剩下的两个概率开始逐步向前进行编码。（10）



6.2.2 霍夫曼编码

霍夫曼编码步骤（例）

2. 从最后一步剩下的两个概率开始逐步向前进行编码。（11）



6.2.2 霍夫曼编码

霍夫曼编码步骤（例）

2. 从最后一步剩下的两个概率开始逐步向前进行编码。（12）

6.2.2 霍夫曼编码

霍夫曼编码步骤（例）

最终编码分配结果如下：

编码平均长度?

$$L = \sum_{i=1}^{N} p(s_i) l(s_i)$$

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								

6.2.2 霍夫曼编码

编码平均长度：

$$L = \sum_{i=1}^{N} p(s_i) l(s_i)$$

$$L_{avg} = 0.4\times1+0.3\times2+0.1\times3+0.1\times4+0.06\times5+0.04\times5$$

$$= 2.2 \text{ bits/pixel}$$

信源熵：

$$H = \sum_{i=1}^{N} - p(s_i)\log_2 p(s_i) \qquad H = 2.14 \text{bits/pixel}$$

6.2.2 霍夫曼编码

解码：

对编码串 010100111100 从左到右扫描：

第一个有效码字为 01010，对应 a_3

第二个有效码字为 011，对应 a_1

…

最终解码结果为 $a_3\ a_1\ a_2\ a_2\ a_6$

符号	编码
a_2	1
a_6	00
a_1	011
a_4	0100
a_3	01010
a_5	01011

6.2.2 霍夫曼编码

霍夫曼编码特点

- 编码本身是一种瞬时、唯一可解码的块编码。

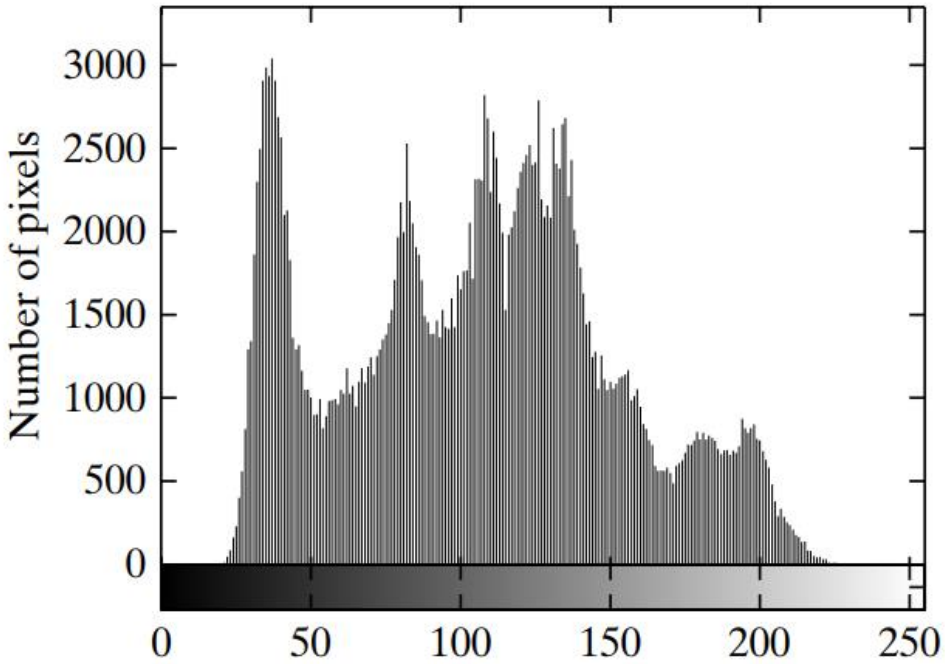
  之所以称为块编码，是因为每个信源符号都映射到了一个固定序列中

- 它是瞬时的，因为编码符号串中的每个码字无需参考后续符号就可以解码

- 它是唯一可解码的，因为任何编码符号串只能以一种方式进行解码

6.2.2 霍夫曼编码

霍夫曼编码在图像压缩中的应用



（a）一副大小为512×512的8比特图像 （b）该图像的直方图

6.2.2 霍夫曼编码

霍夫曼编码在图像压缩中的应用

一幅 512×512 的 8 比特单色图像，灰度不是等概率的。

图像的熵 H = 7.3838bit/pixel

使用的霍夫曼编码平均编码长度 L = 7.428bit/pixel

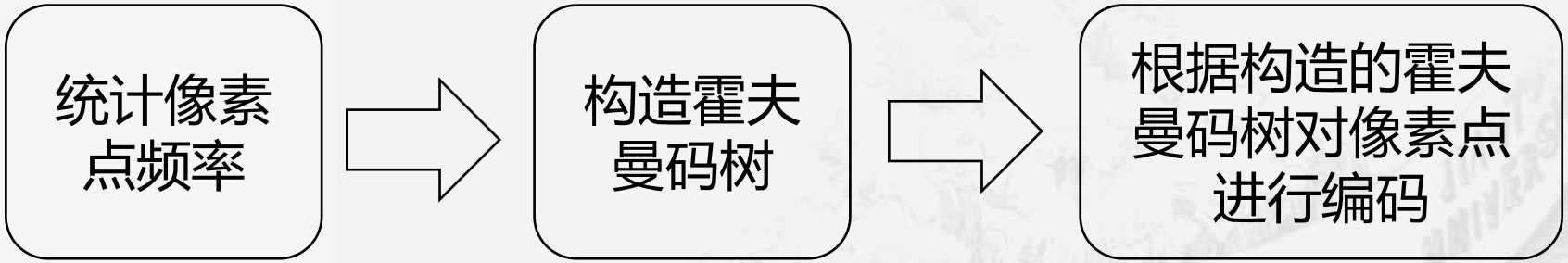
压缩率 C = 8/7.428 = 1.077

相对冗余度 R = 1-（1/1.077）=0.0715

这样使用霍夫曼编码时，原始 8 比特定长灰度表示的 7.15% 就作为编码冗余被去除了。

6.2.2 霍夫曼编码

实践操作：用霍夫曼编码实现图像的编码



参考代码：https://github.com/yhhzsd/Huffman-encoding-and-decoding

6.2.2 霍夫曼编码

霍夫曼编码存在的问题

- 当对大量符号进行编码，构造霍夫曼编码比较复杂
- 对 J 个信源符号，需要进行 J-2 次信源化简和 J-2 次编码分配

有些常用的图像压缩标准，如JPEG和MPEG都规定了默认的霍夫曼编码表。

6.2 基本压缩方法

6.2.3 Golomb编码

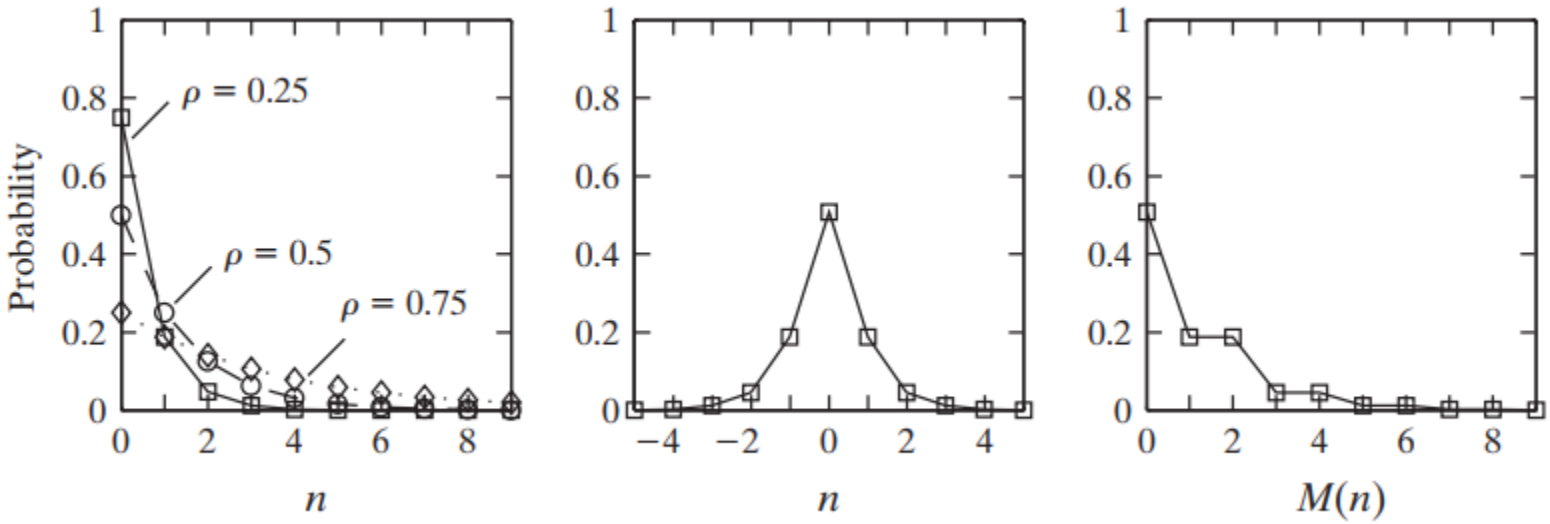
Golomb code是一种**无损的**数据压缩方法，由数学家Solomon W.Golomb 在 1960 年代发明。

- Golomb 编码针对**非负整数**进行编码。
- 符号表中的符号出现的概率符合**几何分布**时，可以取得最优效果，也就是说 Golomb 编码比较适合小的数字比大的数字出现概率高的编码。使用较短的码长编码较小的数字，较长的码长编码较大的数字。

6.2.3 Golomb编码

Golomb 编码效率较高的符号概率分布：

- 几何分布 $\mathrm{P}(n)=(1-\rho)\rho^{n}$，其中 0<$\rho$<1，对应的连续形式是指数分布
- 双边的指数衰减分布



（a）几何分布（b）双边指数分布 （c）重排序后的双边指数分布

6.2.3 Golomb编码

前置知识点：

一元编码（Unary coding）

是一种简单的只能对**非负整数**进行编码的方法，对于任意非负整数 n，它的一元编码就是 n 个 1 后面紧跟着一个 0。例如：

数字	一元编码
0	0
1	10
2	110
3	1110
4	11110
5	111110

6.2.3 Golomb编码

Golomb 编码构建方法

Golomb 编码是一种**分组编码**，需要一个正整数参数 m，然后以 m 为单位对待编码的数字进行分组。

给定一个待编码的非负整数 n 和一个正整数除数 m（m>0），n 关于 m 的 Golomb 编码 $G_m(\mathrm{n})$ 是商 $\lfloor n/m \rfloor$ 的一元编码和 $n \bmod m$ 的二进制表示的合并。

注：符号 $\lfloor x \rfloor$ 表示小于等于 x 的最大整数，$\lceil x \rceil$ 表示大于等于 x 的最小整数，$x \bmod y$ 表示 x 被 y 除的余数。

6.2.3 Golomb编码

$G_m(\mathrm{n})$ 的构建步骤

步骤 1：形成商 $\lfloor n/m \rfloor$ 的一元编码；

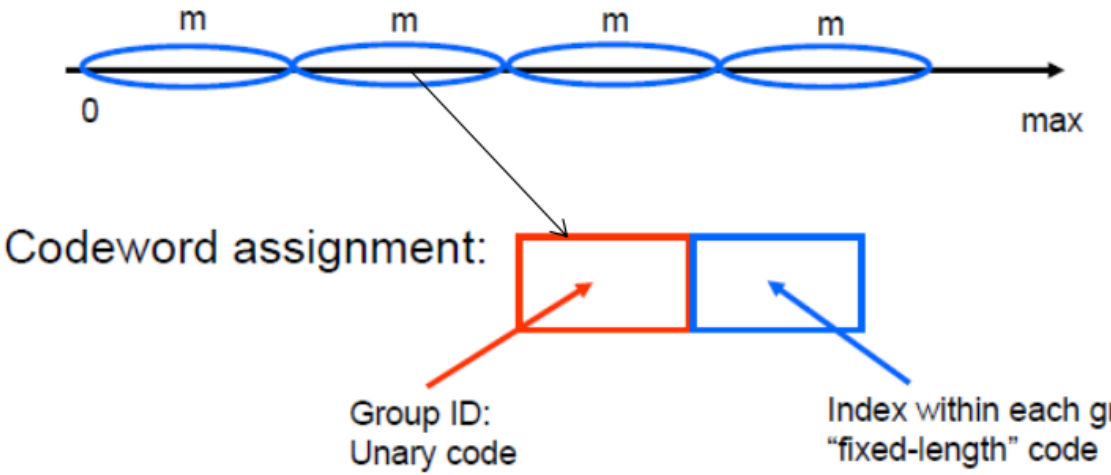
步骤 2：令 $k = \lceil log_2 m \rceil$，$c = 2^k - m$，$r = n \bmod m$，并计算截短的余数 r'，例如，使其满足

$$r' = \begin{cases} r \text{ 截短至 } k-1 \text{ 比特}, & 0 \le r < c \\ r + c \text{ 截短至 } k \text{ 比特}, & \text{其他} \end{cases}$$

步骤 3：连接步骤 1 和步骤 2 的结果。

6.2.3 Golomb编码

Golomb编码就是将编码对象分能成等间隔的若干区间，每个区间有一个索引值：Group Id。



对于Group Id采用二元码编码。

对于区间Group内的编码对象采用定长码。

Golomb Code with m = 5 (Golomb-5)

n	q	r	code
0	0	0	000
1	0	1	001
2	0	2	010
3	0	3	0110
4	0	4	0111

n	q	r	code
5	1	0	1000
6	1	1	1001
7	1	2	1010
8	1	3	10110
9	1	4	10111

n	q	r	code
10	2	0	11000
11	2	1	11001
12	2	2	11010
13	2	3	110110
14	2	4	110111

https://www.cnblogs.com/DwyaneTalk/p/4035206.html

6.2.3 Golomb编码

$$r' = \begin{cases} r \text{ 截短至 } k-1 \text{ 比特}, & 0 \le r < c \\ r+c \text{ 截短至 } k \text{ 比特}, & \text{其他} \end{cases}$$

$G_m(\mathrm{n})$ 的构建步骤（例）

以计算 $G_4(9)$ 为例（n=9，m=4）：

步骤 1：首先求商 $\lfloor 9/4 \rfloor = \lfloor 2.25 \rfloor = 2$ 的一元编码为 110；

步骤 2：令 $k = \lceil log_2 m \rceil = \lceil log_2 4 \rceil = 2$，$c = 2^k - m = 2^2 - 4 = 0$，$\mathrm{r} = n \, mod \, m = 9 \bmod 4 = 1$。

r' 是 $r + c$（即 0001）截短到 2 比特的结果，得到 01；

步骤 3：连接步骤 1 的 110 和步骤 2 的 01，得到 $G_4(9) = 11001$。

6.2.3 Golomb编码

$$r' = \begin{cases} r \text{ 截短至 } k-1 \text{ 比特}, & 0 \le r < c \\ r+c \text{ 截短至 } k \text{ 比特}, & \text{其他} \end{cases}$$

$G_m(\mathrm{n})$ 的构建步骤（例）

以计算 $G_5(14)$ 为例（n=14，m=5）：

步骤 1：首先求商 $\lfloor 14/5 \rfloor = \lfloor 2.8 \rfloor = 2$ 的一元编码为 110；

步骤 2：令 $k = \lceil log_2 m \rceil = \lceil log_2 5 \rceil = 3$，$c = 2^k - m = 2^3 - 5 = 3$，$\mathrm{r} = n \bmod m = 14 \bmod 5 = 4$。

r' 是 $r + c$（即 0111）截短到 3 比特的结果，得到 111；

步骤 3：连接步骤 1 的 110 和步骤 2 的 111，得到 $G_5(14) = 110111$。

6.2.3 Golomb编码

Golomb 码只能用于表示**非负整数**，并且有许多 Golomb 码可供选择，在其有效应用中的一个关键步骤是**参数 m 的选择**。

- 当 $\mathrm{m} = 2^{\mathrm{k}}$，$\mathrm{c} = 0$ 时，对所有的 n，$\mathrm{r}' = \mathrm{r} = n \bmod m$ 截短至 k 比特。此时产生的 Golomb 编码所要求的除法可以用二进制移位操作实现，计算上更加简单。这种编码叫 **Golomb-Rice 码**或 **Rice 码**。
- G_1 编码是非负整数的一元码，因为对所有的 n，有：

$$\lfloor n/1 \rfloor = n，\ n \bmod 1 = 0$$

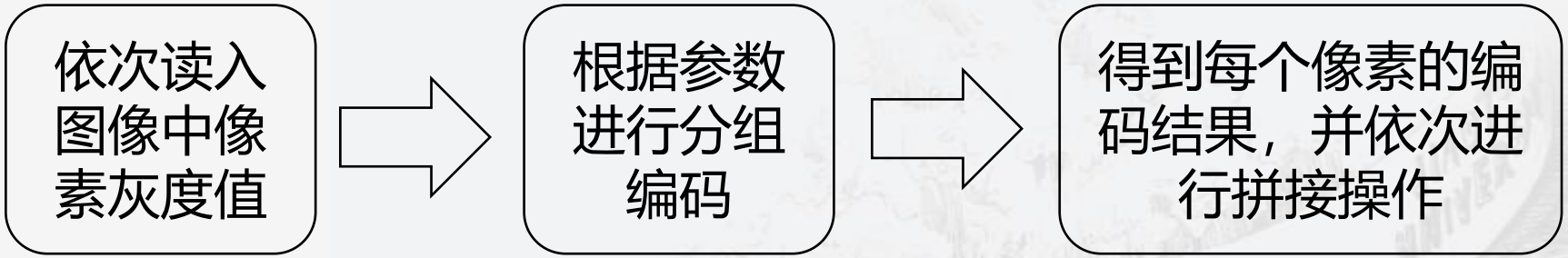
6.2.3 Golomb编码

整数 0-9 的几种 Golomb 码：

n	$G_1(n)$	$G_2(n)$	$G_4(n)$	$G^0_{\exp}(n)$
0	0	00	000	0
1	10	01	001	100
2	110	100	010	101
3	1110	101	011	11000
4	11110	1100	1000	11001
5	111110	1101	1001	11010
6	1111110	11100	1010	11011
7	11111110	11101	1011	1110000
8	111111110	111100	11000	1110001
9	1111111110	111101	11001	1110010

6.2.3 Golomb编码

实践操作：用Golomb编码实现图像的编码



参考代码：https://github.com/brookicv/GolombCode

6.2 基本压缩方法

6.2.4 算术编码

为什么提出算术编码?

以信源字符序列 {x，y} 为例:

设字符序列 {x，y} 对应的概率为 {1/3，2/3}，使用霍夫曼编码时，{x，y} 的码字分别为 0 和 1，也就是两个符号信息的编码长度都为 1。对于出现概率大的字符 y 并未能赋予较短的码字。

为了提高编码效率，Elias 等人提出了算术编码算法。

6.2.4 算术编码

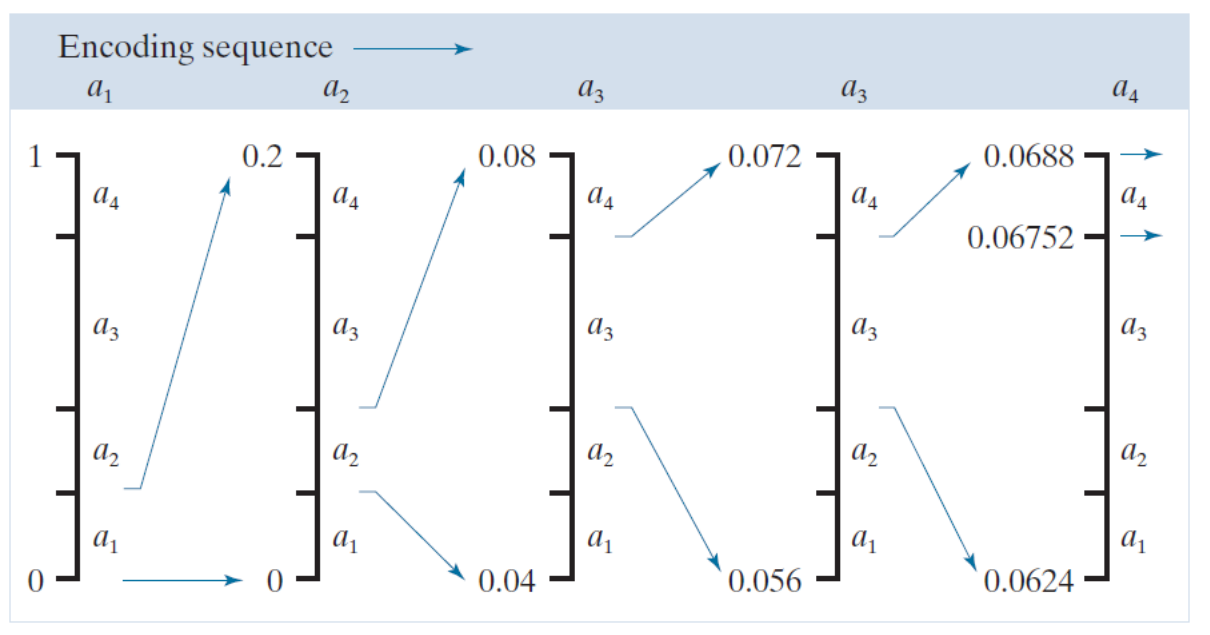
算术编码原理

- 算术编码为信源符号的整个序列分配了一个单一的算数码字，该码字本身定义了一个介于 0 和 1 之间的实数间隔，即小数区间。消息越长，编码表示它的间隔就越小。
- 以小数表示间隔，表示的间隔越小所需的二进制位数就越多，码字就越长。反之，间隔越大，编码所需的二进制位数就少，码字就短。

6.2.4 算术编码

算术编码原理

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)



算术编码示例

设图像信源编码可用 a、b、c、d 这 4 个符号来表示，图像信源字符集为 {dacba}，信源字符出现的概率分别如下表所示，采用算术编码对图像字符集编码。

信源字符	a	b	c	d
出现概率	0.4	0.2	0.2	0.2

算术编码示例（续）

(1) 根据已知条件和数据可知，信源各字符在区间 [0，1] 内的子区间间隔分别如下：

a=[0.0 ， 0.4)　　b=[0.4 ， 0.6)

c=[0.6 ， 0.8)　　d=[0.8 ， 1.0)

算术编码示例（续）

(2) 计算中按如下公式产生新的子区间：

$$\begin{cases} Start_N = Start_B + Left_C \times L \\ End_N = Start_B + Right_C \times L \end{cases}$$

其中：

$Start_N$为新区间左端点，End_N为新区间右端点

$Start_B$为前一字符区间左端点，L为前一字符区间长度

$Left_C$为当前字符区间左端点，$Right_C$为当前字符区间右端点

6.2.4 算术编码

$$\begin{cases} Start_N = Start_B + Left_C \times L \\ End_N = Start_B + Right_C \times L \end{cases}$$

算术编码示例（续）

{dacba}

a=[0.0，0.4)　　b=[0.4，0.6)

c=[0.6，0.8)　　d=[0.8，1.0)

(3) 第1个被压缩的字符为“d”，其初始子区间为 [0.8，1.0)

(4) 第2个被压缩的字符为“a”，由于其前面的字符取值区间为 [0.8，1.0)，因此，字符“a”应在前一字符区间间隔[0.8，1.0) 的 [0.0，0.4) 子区间内，根据公式可得：

$Start_N$ =0.8+0.0×(1.0-0.8)=0.8

End_N =0.8+0.4×(1.0-0.8)=0.88

6.2.4 算术编码

$$\begin{cases} Start_N = Start_B + Left_C \times L \\ End_N = Start_B + Right_C \times L \end{cases}$$

算术编码示例（续）

{dacba}

a=[0.0 ， 0.4)　　b=[0.4 ， 0.6)

c=[0.6 ， 0.8)　　d=[0.8 ， 1.0)

(5) 第3个被压缩的字符为“c”，由于其前面的字符取值区间为 [0.8 ， 0.88)，因此，字符“c”应在前一字符区间间隔[0.8 ， 0.88) 的 [0.6 ， 0.8) 子区间内，根据公式可得：

$Start_N$ =0.8+0.6×(0.88-0.8)=0.848

End_N =0.8+0.8×(0.88-0.8)=0.864

6.2.4 算术编码

$$\begin{cases} Start_N = Start_B + Left_C \times L \\ End_N = Start_B + Right_C \times L \end{cases}$$

算术编码示例（续）

{dacba}

a=[0.0 ， 0.4)　　b=[0.4 ， 0.6)

c=[0.6 ， 0.8)　　d=[0.8 ， 1.0)

(6) 第4个被压缩的字符为“b”，由于其前面的字符取值区间为 [0.848 ， 0.864)，因此，字符“b”应在前一字符区间间隔 [0.848 ， 0.864) 的 [0.4 ， 0.6) 子区间内，根据公式可得：

$Start_N$ =0.848+0.4×(0.864-0.848)=0.8544

End_N =0.848+0.6×(0.864-0.848)=0.8576

6.2.4 算术编码

$$\begin{cases} Start_N = Start_B + Left_C \times L \\ End_N = Start_B + Right_C \times L \end{cases}$$

算术编码示例（续）

{dacba}

a=[0.0 ， 0.4)　　b=[0.4 ， 0.6)

c=[0.6 ， 0.8)　　d=[0.8 ， 1.0)

(7) 第5个被压缩的字符为“a”，由于其前面的字符取值区间为 [0.8544 ， 0.8576)，因此，字符“a”应在前一字符区间间隔 [0.8544 ， 0.8576) 的 [0.0 ， 0.4) 子区间内，根据公式可得：

$Start_N$ =0.8544+0.0×(0.8576-0.8544)=0.8544

End_N =0.8544+0.4×(0.8576-0.86544)=0.85568

6.2.4 算术编码

算术编码示例（续）

经过上述计算，字符集 {dacba} 被描述在实数子区间 [0.8544, 0.85568) 内，即该区间内的任一实数值都唯一对应该字符序列 {dacba}。

在[0.8544, 0.85568) 子区间内的最短二进制代码为 0.11011011，去掉小数点及其前的字符，从而得到该字符序列的算术编码为 11011011。平均码字长度为：$\frac{8}{5} = 1.6$ 比特/字符。

https://www.sojson.com/hexconvert.html

0.8544= 0.11011010101110011111010101011001101100111110100001

0.85568=0.11011011000011011101100000101111110101110101111000101

6.2.4 算术编码

经过上述计算，字符集 {dacba} 被描述在实数子区间 [0.8544, 0.85568) 内。

思考：如何解码?

d: [0.8 ， 1.0)

da: [0.8 ， 0.88)

dac: [0.848 ， 0.864)

dacb: [0.8544 ， 0.8576)

dacba: [0.8544, 0.85568)

a=[0.0 ， 0.4)

b=[0.4 ， 0.6)

c=[0.6 ， 0.8)

d=[0.8 ， 1.0)

https://blog.csdn.net/qq_36752072/article/details/77986159

6.2.4 算术编码

算术编码的特点

- 算术编码是信息保持型编码，它不像霍夫曼编码，无需为一个符号设定一个码字；
- 当信源字符出现的概率比较接近时，算术编码效率高于霍夫曼编码的效率，在图像通信中常用它来取代霍夫曼编码；
- 实现算术编码算法的硬件比霍夫曼编码复杂。

6.2.4 算术编码

当被编码的序列长度增加时，得到的算术编码接近香农第一定理所设定的界限。实际上**有两个因素会使得编码性能无法达到这一界限**：

- 为了将一个消息与其他消息分开，增加了**消息结束指示符**
- 所用的**算法精度是有限的**

6.2 基本压缩方法

6.2.1 压缩编码方法分类

6.2.2 霍夫曼编码

6.2.3 Golomb编码

6.2.4 算术编码

6.2.5 LZW编码

6.2.5 LZW编码

霍夫曼编码，Golomb 编码和算数编码都集中在消除**编码冗余**上，而 Lemple-Ziv-Welch（LZW）编码技术致力于消除图像中的**空间冗余**。

LZW 编码**将定长码字分配给变长信源符号序列**。它的关键特点是**它不需要事先知道被编码符号出现的概率**。使用 LZW 压缩技术的文件格式包括 GIF，TIFF 和 PDF 等。

6.2.5 LZW编码

75	75	77	77	78	78	78	78
75	75	77	77	78	78	78	78
75	75	77	77	78	78	78	78
75	75	77	77	78	78	78	78
75	75	77	77	78	78	78	78
75	75	75	77	77	77	77	77
74	74	74	74	75	75	75	75
73	73	73	73	73	73	73	73
72	72	72	72	72	72	72	72
72	72	70	70	72	72	72	72
72	72	70	70	72	72	72	72
72	72	70	70	72	72	72	72
72	72	70	70	72	72	72	72
72	72	70	70	72	72	72	72
72	72	70	70	72	72	72	73
73	73	71	71	71	73	74	76
73	73	71	71	71	73	74	76
73	73	71	71	71	71	73	74

6.2.5 LZW编码

LZW 编码方法

构建一个包含被编码信源符号的码书或字典（*例如，对于 8 比特单色图像，字典中的前 256 个字被分配给灰度值 0，1，2，…，255*）。

当编码器顺序分析图像像素时，不在字典中的灰度序列被放置到字典的下一个未用位置（*例如，如果图像前两个相邻像素为白色，则序列“255-255”就可能被分配到字典的 256 位置*），当下一次再遇到该灰度序列时，就用字典的已有码字来表示，这样就去掉了冗余信息。

6.2.5 LZW编码

LZW 编码示例

一个 4×4 大小的 8 比特图像：

39	39	126	126
39	39	126	126
39	39	126	126
39	39	126	126

将图像按照从左到右、从上到下的顺序处理其像素。

一个 512 字的字典：

字典位置	条　目
0	0
1	1
…	…
255	255
256	-
…	…
511	-

LZW 编码示例（续）

编码过程：

（1）将图像中每个灰度值与“当前可识别序列”中的变量连接；

（2）在字典词条中搜索第一步中得到的结果序列。若存在，则用位于字典中的词条代替它，此时既不产生输出代码，也不更改字典；若不存在，则将“当前可识别序列”的字典位置作为下一个编码值输出，并将该不可识别的序列添加到字典词条中，正被处理的像素值作为下一个“当前可识别序列”的值。

6.2.5 LZW编码

LZW 编码示例（续）

处理图像第一行：

39	39	126	126
39	39	126	126
39	39	126	126
39	39	126	126

当前可识别的序列	正被处理的像素	编码后的输出	字典位置（码字）	字典词条
存在	39			
不存在 39	39	39	256	39-39
不存在 39	126	39	257	39-126
不存在 126	126	126	258	126-126

6.2.5 LZW编码

LZW 编码示例（续）

处理图像第二行：

39	39	126	126
39	39	126	126
39	39	126	126
39	39	126	126

	当前可识别的序列	正被处理的像素	编码后的输出	字典位置（码字）	字典词条
不存在	126	39	126	259	126-39
存在	39	39			
不存在	39-39	126	256	260	39-39-126
存在	126	126			

6.2.5 LZW编码

Currently Recognized Sequence	Pixel Being Processed	Encoded Output	Dictionary Location (Code Word)	Dictionary Entry
	39			
39	39	39	256	39-39
39	126	39	257	39-126
126	126	126	258	126-126
126	39	126	259	126-39
39	39			
39-39	126	256	260	39-39-126
126	126			
126-126	39	258	261	126-126-39
39	39			
39-39	126			
39-39-126	126	260	262	39-39-126-126
126	39			
126-39	39	259	263	126-39-39
39	126			
39-126	126	257	264	39-126-126
126		126		

TABLE 8.7 LZW coding example.

LZW 编码示例（续）

编码前：4×4×8 = 128 bit

编码后：10×9 = 90 bit

压缩率：C = 128：90 = 1.42：1

将编码后的输出从上到下读取，就是压缩编码结果。即39，39，126，126，256，258，260，259，257，126。解码时，根据压缩编码结果，字典位置和字典词条即可恢复图像。

6.2.5 LZW编码

LZW 编码在图像压缩中的应用

考虑 512×512 的 8 比特图像，Adobe Photoshop 的未压缩 TIFF 图像需要 286,740 字节空间（头文件 24,596字节）。使用 TIFF 的 LZW 编码压缩后，文件大小为 224,420 字节。压缩率 C=1.28（霍夫曼编码压缩率为 C=1.077）。**LZW编码实现的额外压缩是由于消除了图像的某些空间冗余。**

6.2.5 LZW编码

LZW 编码特点

- 编码字典是在对数据进行编码的同时创建的
- 解码器对编码数据流进行解码的同时建立一个同样的解压缩字典
- 实际应用中需要处理字典溢出问题

6.2.5 LZW编码

LZW 编码处理字典溢出的策略

- 简单的解决方法：当字典已满时，使用一个新初始化后的字典继续编码；
- 复杂的解决办法：监控压缩性能，并在性能变得低下或不可接受时刷新字典；
- 需要时可跟踪并替换那些使用最少的字典词条。

西安交通大學
XI'AN JIAOTONG UNIVERSITY

本章结束