



Research Institute for Future Media Computing Institute of Computer Vision
未来媒体技术与研究所 计算机视觉研究所



多媒体系统导论

Fundamentals of Multimedia System

授课教师：朱映映教授

Email: zhuyy@szu.edu.cn

第七讲

Lossless Compression Algorithms

第7章

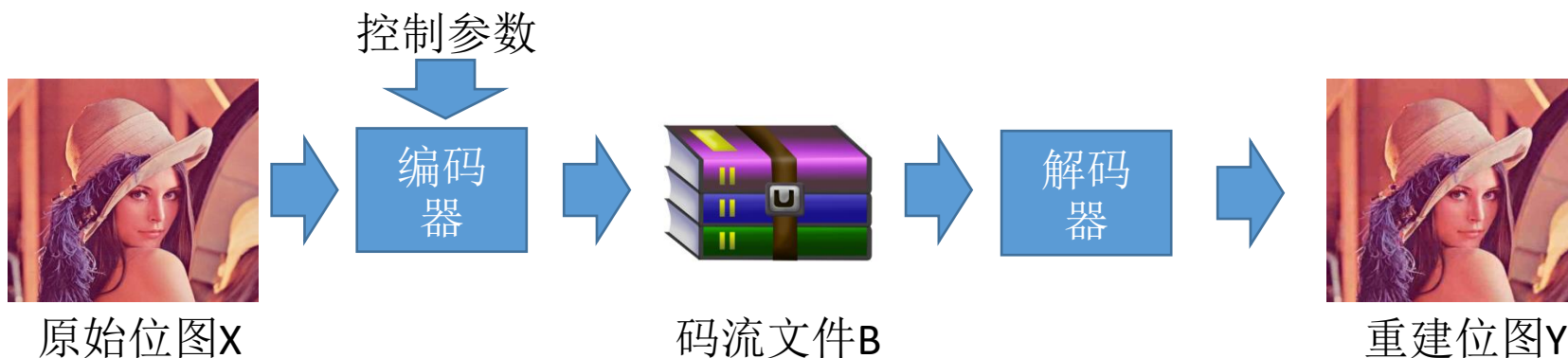
0. 图像压缩

- ◆ 图像压缩是数据压缩技术在数字图像上的应用，它的目的是减少图像数据中的冗余信息从而用更加高效的格式存储和传输数据。
- ◆ 根据解码后的图像数据是否与原始图像数据一致，可分为有损图像压缩和无损图像压缩
- ◆ 常用的技术包括：
 - 预测编码：如差分预测
 - 变换编码：如DCT/IDCT
 - 熵编码：如Huffman coding，算术编码等
 - 量化
 - 色度二次采样

0. 图像压缩(续)

◆ 图像压缩系统包括一个编码器和相应的解码器

- 编码器：读入待压缩的位图图像数据，通过各种压缩技术，以消除图像数据中存在的空间、统计、视觉等冗余。编码器输出的**二进制码流文件**的尺寸由用户设定的参数（如量化等级）控制。
- 解码器：读入**二进制码流文件**并解码，重建位图图像数据。
- **二进制码流文件**：包含了图像的完整信息，如分辨率，通道数，颜色空间模型，色度二次采样格式，量化参数，Huffman tree等其他一些解码中需要用到的信息



1. 数据无损压缩概述

◆ 数据可被压缩的依据

- 数据本身存在冗余
- 听觉系统的敏感度有限
- 视觉系统的敏感度有限

◆ 三种多媒体数据类型

- 文字 (text)数据——无损压缩
 - 根据数据本身的冗余(Based on data redundancy)
- 声音(audio)数据——有损压缩
 - 根据数据本身的冗余(Based on data redundancy)
 - 根据人的听觉系统特性(Based on human hearing system)
- 图像(image)/视像(video) 数据——有损压缩
 - 根据数据本身的冗余(Based on data redundancy)
 - 根据人的视觉系统特性(Based on human visual system)

1. 数据无损压缩概述(续1)

◆ 数据无损压缩的理论——信息论(information theory)

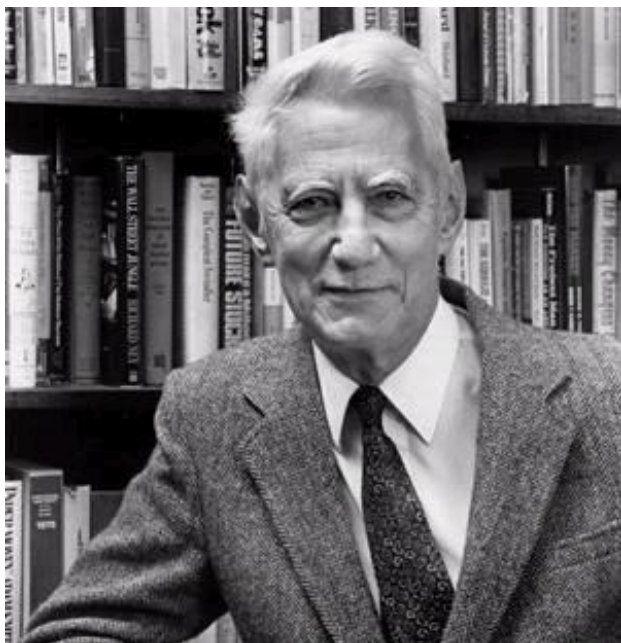
- 1948年创建的数学理论的分支学科，研究信息的编码、传输和存储
- 该术语源于Claude Shannon (香农)发表的“A Mathematical Theory of Communication” 论文题目，提议用二进制数据对信息进行编码
- 最初只应用于通信工程领域，后来扩展到包括计算在内的其他多个领域，如信息的存储、信息的检索等。在通信方面，主要研究数据量、传输速率、信道容量、传输正确率等问题。

◆ 数据无损压缩的方法

- 霍夫曼编码(Huffman Coding)
- 算术编码(Arithmetic Coding)
- 行程长度编码(Run-Length Coding)
- 词典编码(Dictionary Coding)
-

1. 数据无损压缩概述(续2)

◆ 信息论之父介绍



- ◆ The Father of Information Theory—Claude Elwood Shannon
 - Born: 30 April 1916 in Gaylord, Michigan, USA
 - Died: 24 Feb 2001 in Medford, Massachusetts, USA

2. 数据的冗余

◆ 冗余概念

- 人为冗余

- 在信息处理系统中，使用两台计算机做同样的工作是提高系统可靠性的一种措施
- 在数据存储和传输中，为了检测和恢复在数据存储或数据传输过程中出现的错误，根据使用算法的要求，在数据存储或数据传输之前把额外的数据添加到用户数据中，这个额外的数据就是冗余数据

- 视听冗余

- 由于人的视觉系统和听觉系统的局限性，在图像数据和声音数据中，有些数据确实是多余的，使用算法将其去掉后并不会丢失实质性的信息或含义，对理解数据表达的信息几乎没有影响

- 数据冗余

- 不考虑数据来源时，单纯数据集中也可能存在多余的数据，去掉这些多余数据并不会丢失任何信息，这种冗余称为数据冗余，而且还可定量表达

2. 数据的冗余(续1)

◆ 熵(entropy)

- 按照香农(Shannon)的理论, 在有限的互斥和联合穷举事件的集合中, 熵为事件的信息量的平均值, 也称事件的平均信息量(mean information content), 数学表示为

$$H(X) = \sum_{i=1}^n h(x_i) = \sum_{i=1}^n p(x_i) I(x_i) = - \sum_{i=1}^n p(x_i) \log_2 p(x_i) \quad (3-3)$$

其中, (1) $X = \{x_1, x_2, \dots, x_n\}$ 是事件 $x_i (i = 1, 2, \dots, n)$ 的集合, 并满足 $\sum_{i=1}^n p(x_i) = 1$;

(2) $I(x_i) = -\log_2 p(x_i)$ 表示某个事件 x_i 的信息量, 其中 $p(x_i)$ 为事件 x_i 出现的概率, $0 < p(x_i) \leq 1$; $h(x_i) = -p(x_i) \log_2 p(x_i)$ 表示事件 x_i 的熵。

【例】 $X = \{a, b, c\}$ 是由 3 个符号构成的集合, 符号 a, b 和 c 出现的概率分别为 $p(a) = 0.5$, $p(b) = 0.25$, $p(c) = 0.25$, 那么符号 a, b 和 c 的熵分别等于 0.5, 0.5, 0.5, 这个集合的熵为:

$$H(X) = p(a) I(a) + p(b) I(b) + p(c) I(c) = 1.5 \quad (\text{Sh})$$

2. 数据的冗余(续2)

- ◆ 熵代表了信源 X 中每个符号所需要的最小平均位数。
- ◆ 对信源 X 中每个符号进行编码所需的平均位数的下界。
- ◆ 编码方案旨在尽可能地接近熵。

压缩比与编码效率

◆ Compression Ratio

- 设 B_0 为压缩前表示某信源所需的总位数, B_1 为压缩后所需的总位数, 则

$$\text{Compression Ratio (压缩比或压缩率)} = B_0 / B_1$$

- 一般而言, 期望压缩比大于1, 如果压缩比越高, 意味着无损压缩方案越好

◆ 编码效率 η

- $\eta = H / L$, L 为编码后的平均码长
- 编码效率越接近于1, 意味着编码方案越好

3. 统计编码

◆ 统计编码（熵编码）

- 给已知统计信息的符号分配代码的数据无损压缩方法

◆ 编码方法

- 香农-范诺编码
- 霍夫曼编码
- 算术编码

◆ 编码特性

- 香农-范诺编码和霍夫曼编码的原理相同，都是根据符号集中各个符号出现的频繁程度来编码，出现次数越多的符号，给它分配的代码位数越少
- 算术编码使用0和1之间的实数的间隔长度代表概率大小，概率越大间隔越长，编码效率可接近于熵

3.1 统计编码—香农-范诺编码

◆ 香农-范诺编码(Shannon–Fano Coding)

- 在香农的源编码理论中，熵的大小表示非冗余的不可压缩的信息量
- 在计算熵时，如果对数的底数用2，熵的单位就用“香农(Sh)”，也称“位(bit)”。 “位”是1948年Shannon首次使用的术语。例如

事件 x_i 的熵 $h(x_i) = -p(x_i) \log_2(1/p(x_i))$ ，

表示编码符号 x_i 所需要的位数

- 最早阐述和实现“从上到下”的熵编码方法的人是Shannon(1948年)和Fano(1949年)，因此称为香农-范诺(Shannon–Fano)编码法

3.1 香农-范诺编码

◆ 香农-范诺编码举例

- 有一幅40个像素组成的灰度图像，灰度共有5级，分别用符号A, B, C, D和E表示。40个像素中出现灰度A的像素数有15个，出现灰度B的像素数有7个，出现灰度C的像素数有7个，其余情况见表3-1
 - (1) 计算该图像可能获得的压缩比的理论值
 - (2) 对5个符号进行编码
 - (3) 计算该图像用香农范诺编码后的压缩比的实际值

表 3-1 符号在图像中出现的数目

符 号	A	B	C	D	E
出现的次数	15	7	7	6	5
出现的概率	$\frac{15}{40}$	$\frac{7}{40}$	$\frac{7}{40}$	$\frac{6}{40}$	$\frac{5}{40}$

3.2.1 香农-范诺编码(续1)

(1) 压缩比的理论值

按照常规的编码方法，表示5个符号最少需要3位，如用000表示A，001表示B，...，100表示E，其余3个代码(101, 110, 111)不用。这就意味每个像素用3位，编码这幅图像总共需要120位。按照香农理论，这幅图像的熵为

$$\begin{aligned} H(X) &= -\sum_{i=1}^n p(x_i) \log_2 p(x_i) \\ &= -p(A) \log_2(p(A)) - p(B) \log_2(p(B)) - \cdots - p(E) \log_2(p(E)) \\ &= (15/40) \log_2(40/15) + (7/40) \log_2(40/7) + \cdots + (5/40) \log_2(40/5) \approx 2.196 \end{aligned}$$

这个数值表明，每个符号不需要用3位构成的代码表示，而用2.196位就可以，因此40个像素只需用87.84位就可以，因此在理论上，这幅图像的的压缩比为 $120 : 87.84 \approx 1.37:1$ ，实际上就是 $3 : 2.196 \approx 1.37 : 1$

3.1 香农-范诺编码(续2)

(2) 符号编码

对每个符号进行编码时采用“**从上到下**”的方法。首先按照符号出现的频度或概率降序排序，如A，B，C，D和E，见3-2。然后使用递归方法分成两个部分，每一部分具有近似相同的次数，如图3-1所示

表 3-2 Shannon-Fano 算法举例

符号	出现的次数($p(x_i)$)	$\log_2(1/p(x_i))$	分配的代码	需要的位数
A	15 (0.375)	1.4150	00	30
B	7 (0.175)	2.5145	01	14
C	7 (0.175)	2.5145	10	14
D	6 (0.150)	2.7369	110	18
E	5 (0.125)	3.0000	111	15

3.1 香农-范诺编码(续3)

(3) 算法步骤

- ① 对于一个给定的符号列表，制定了概率相应的列表或频率计数，使每个符号的相对发生频率是已知。
- ② 根据频率降序排列，频率高的符号在左边，频率低的符号在右边。
- ③ 分为两部分，使左边部分的总频率和尽可能接近右边部分的总频率和。
- ④ 该列表的左半边分配二进制数字0，右半边是分配的数字1。这意味着，在第一半符号代都是将所有从0开始，第二半的代码都从1开始。
- ⑤ 对左、右半部分递归应用步骤3和4，细分群体，并添加位的代码，直到每个符号已成为一个相应的代码树的叶。

3.1 香农-范诺编码(续4)

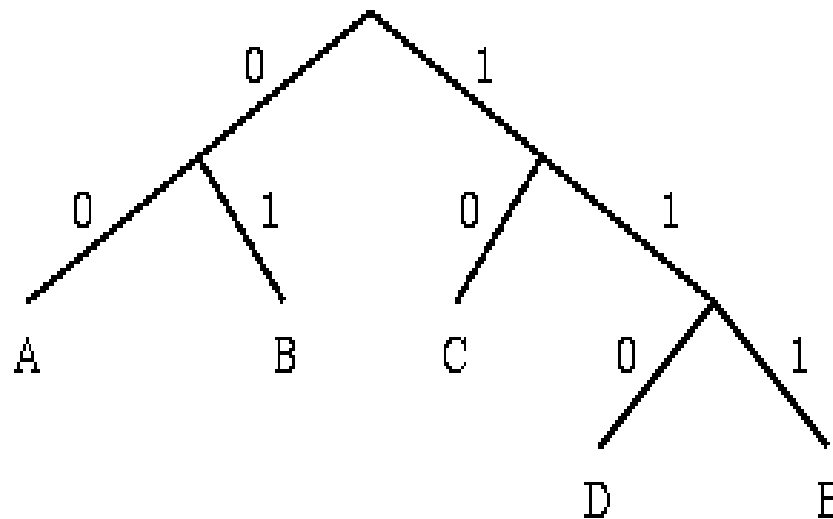
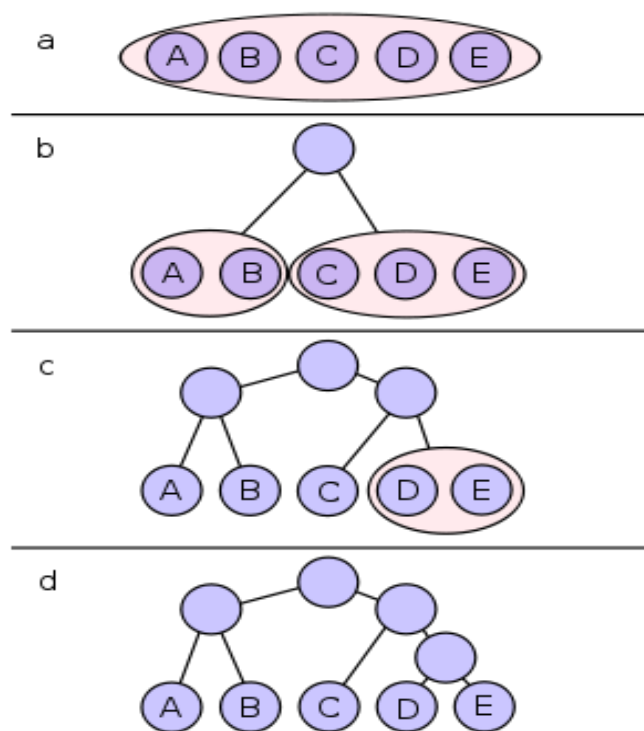


图3-1 香农-范诺算法编码举例

(4) 压缩比的实际值

按照这种方法进行编码需要的总位数为 $30+14+14+18+15=91$ ，实际的压缩比为 $120:91 \approx 1.32:1$

3.2 统计编码——霍夫曼编码

◆ 霍夫曼编码(Huffman Coding)

- 霍夫曼(D.A. Huffman)在1952年提出和描述的“**从下到上**”的熵编码方法
- 根据给定数据集中各元素所出现的频率来压缩数据的一种统计压缩编码方法。这些元素(如字母)出现的次数越多，其编码的位数就越少
- 广泛用在JPEG, MPEG, H.26X等编码标准中

3.2 霍夫曼编码— Case Study 1

◆ 霍夫曼编码举例1

- 现有一个由5个不同符号组成的**30**个符号的字符串：
BABACACADADABBCBABEBEDDABEEEEBB
- 计算
 - (1) 该字符串的霍夫曼码
 - (2) 该字符串的熵
 - (3) 该字符串的平均码长
 - (4) 编码前后的压缩比

3.2 霍夫曼编码— Case Study 1 (续1)

符号出现的概率

符号	出现的次数	$\log_2(1/p_i)$	分配的代码	需要的位数
B	10	1.585	?	
A	8	1.907	?	
C	3	3.322	?	
D	4	2.907	?	
E	5	2.585	?	
合计	30			

3.2.2 霍夫曼编码— Case Study 1 (续2)

(1) 计算该字符串的霍夫曼码

步骤①：按照符号出现概率大小的顺序对符号进行排序

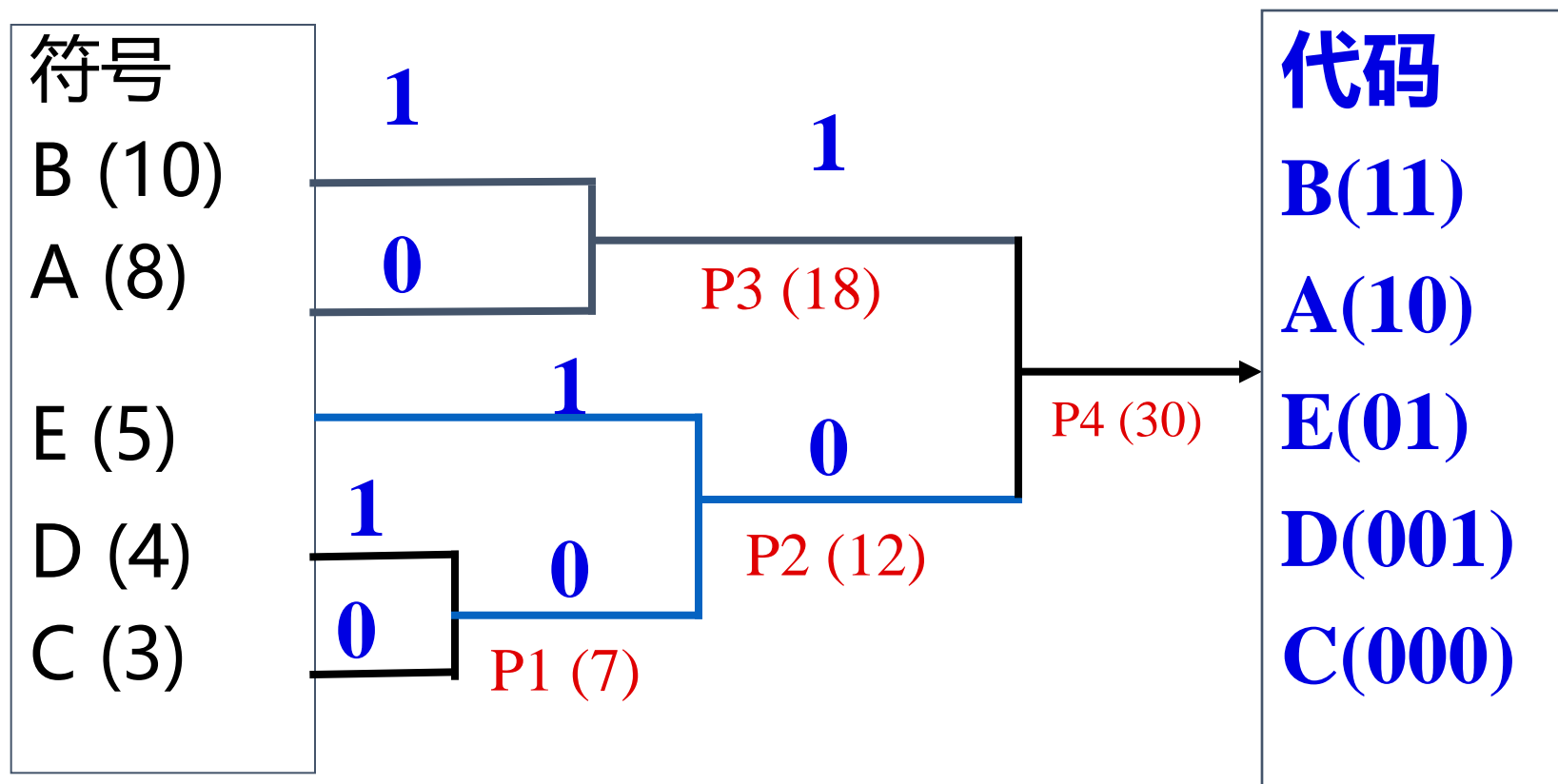
步骤②：把概率最小的两个符号组成一个节点P1

步骤③：重复步骤②，得到节点P2, P3, P4,, PN, 形成一棵树，其中的PN称为根节点

步骤④：从根节点PN开始到每个符号的树叶，从上到下标上0(上枝)和1(下枝)，至于哪个为1哪个为0则无关紧要，但通常把概率大的标成1，概率小的标成0

步骤⑤：从根节点PN开始顺着树枝到每个叶子分别写出每个符号的代码

3.2 霍夫曼编码— Case Study 1 (续3)



3.2 霍夫曼编码— Case Study 1 (续4)

5个符号的代码

符号	出现的次数	$\log_2(1/p_i)$	分配的代码	需要的位数
B	10	1.585	11	20
A	8	1.907	10	16
C	3	3.322	000	9
D	4	2.907	001	12
E	5	2.585	01	10
合计	30	1.0		67

30个字符组成的字符串需要67位

3.2 霍夫曼编码— Case Study 1 (续5)

(2) 计算该字符串的熵

其中, $X = \{x_1, \dots, x_n\}$ 是事件 $x_i (i = 1, 2, \dots, n)$ 的集合,

并满足 $\sum_{i=1}^n p(x_i) = 1$ $H(X) = \sum_{i=1}^n p(x_i) I(x_i) = -\sum_{i=1}^n p(x_i) \log_2 p(x_i)$

➤ $H(S)$

$$\begin{aligned} &= (8/30) \times \log_2(30/8) + (10/30) \times \log_2(30/10) + \\ &\quad (3/30) \times \log_2(30/3) + (4/30) \times \log_2(30/4) + \\ &\quad (5/30) \times \log_2(30/5) \\ &= [30 \lg 30 - (8 \times \lg 8 + 10 \times \lg 10 + 3 \times \lg 3 + 4 \\ &\quad \times \lg 4 + 5 \lg 5)] / (30 \times \log_2 2) \\ &= (44.3136 - 24.5592) / 9.0308 = 2.1874 \text{ (Sh)} \end{aligned}$$

3.2 霍夫曼编码— Case Study 1 (续6)

(3) 计算该字符串的平均码长

$$\text{平均码长: } l = \sum_{i=1}^N l_i p(l_i)$$

$$= (2 \times 8 + 2 \times 10 + 3 \times 3 + 3 \times 4 + 2 \times 5) / 30$$

$$= 2.233 \text{ 位/符号}$$

$$\text{平均码长: } 67/30 = 2.233 \text{ 位}$$

(4) 计算编码前后的压缩比

- 编码前: 5个符号需3位, 30个字符, 需要90位
- 编码后: 共67位

$$\text{压缩比: } 90/67 = 3/2.233 = 1.34:1$$

3.2 霍夫曼编码— Case Study 2

◆ 霍夫曼编码举例2

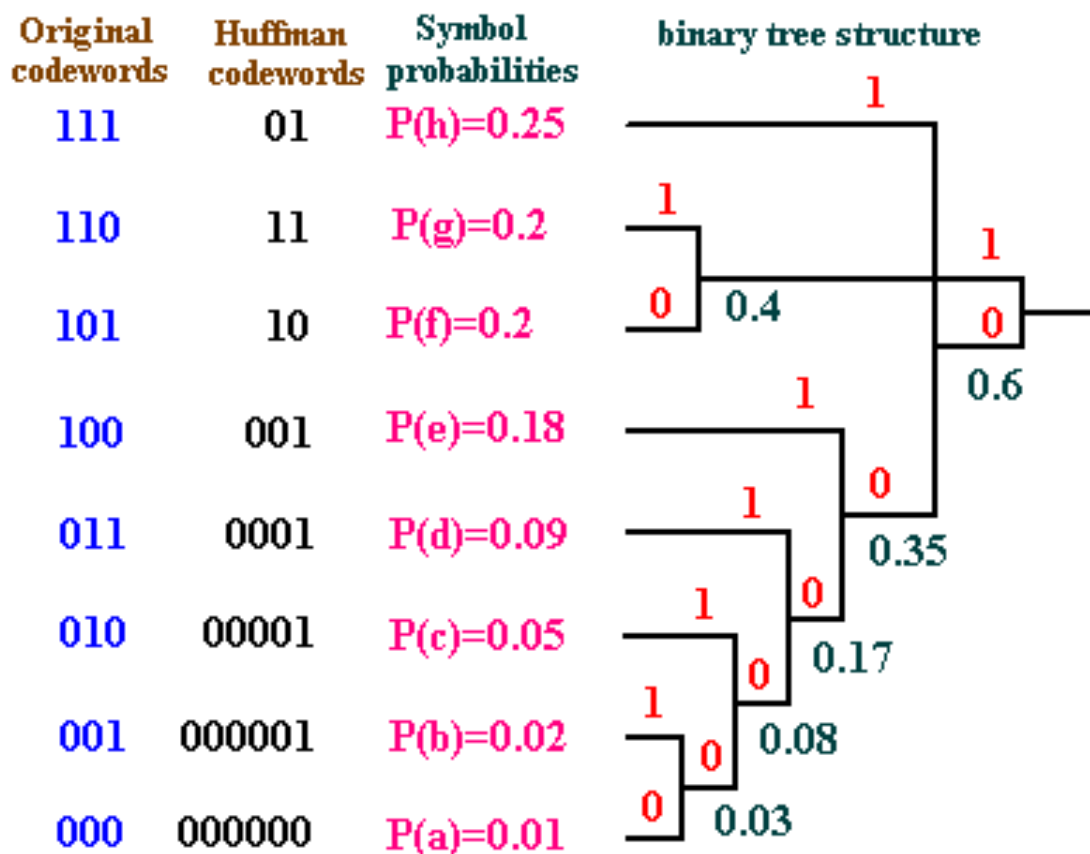
- 编码前

- $N = 8$ symbols: {a,b,c,d,e,f,g,h},
- 3 bits per symbol ($N = 2^3 = 8$)
- $P(a) = 0.01, P(b)=0.02, P(c)=0.05, P(d)=0.09, P(e)=0.18,$
 $P(f)=0.2, P(g)=0.2, P(h)=0.25$

- 计算

- (1) 该字符串的霍夫曼码
- (2) 该字符串的熵
- (3) 该字符串的平均码长
- (4) 编码效率

3.2 霍夫曼编码— Case Study 2 (续1)



3.2 霍夫曼编码— Case Study 2 (续2)

Average length per symbol (before coding):

$$\bar{L} = \sum_{i=1}^8 3P(i) = 3 \text{ bits/symbol}$$

(2) Entropy:

$$H = -\sum_{i=1}^8 P(i) \log_2 P(i) = 2.5821 \text{ bits/symbol}$$

(3) Average length per symbol (with Huffman coding):

$$\bar{L}_{Huf} = 2.63 \text{ bits/symbol}$$

(4) Efficiency of the code:

$$H / \bar{L}_{Huf} = 98\%$$

3.2 霍夫曼编码

◆ 霍夫曼编码的重要性质：

1. 唯一前缀性。任何一个霍夫曼编码都不能作为另一个霍夫曼编码的前缀
2. 平均码字最短, 编码依赖于信源统计特性
 - 1) 两个频率最低的字符具有相同长度的编码
 - 2) 频率高的符号码字比出现频率低的符合的码字短
 - 3) 信源 X 的平均编码长度严格小于 $H(X)+1$
 - 4) 编码的信源概率是2的负幂时, 效率达到100%; 但是对等概率分布的信源情况, 产生定长码, 效率较低
3. 码字长短不一, 实时硬件实现复杂, 特别是译码
4. 编出的码不唯一
 - 1) 为两个概率分配码字的0和1可以任意分配。
 - 2) 在排序过程中, 相等概率的顺序是随机的。

3.3 RLE编码

◆ 行程长度编码(Run-Length Coding)

- ✓ 一种无损压缩数据编码技术，它利用重复的数据单元有相同的数值这一特点对数据进行压缩。对相同的数值只编码一次，同时计算出相同值重复出现的次数。
- ✓ 对某些相同灰度级成片连续出现的图像，特别是二值图像，是一种高效的编码方法。
- ✓ 在JPEG，MPEG，H.261和H.263等压缩方法中，RLE用来对图像数据变换和量化后的系数进行编码
- ✓ 例：假设有一幅灰度图像第n行的像素值如图3-2所示。用RLE编码方法得到的代码为：80315084180

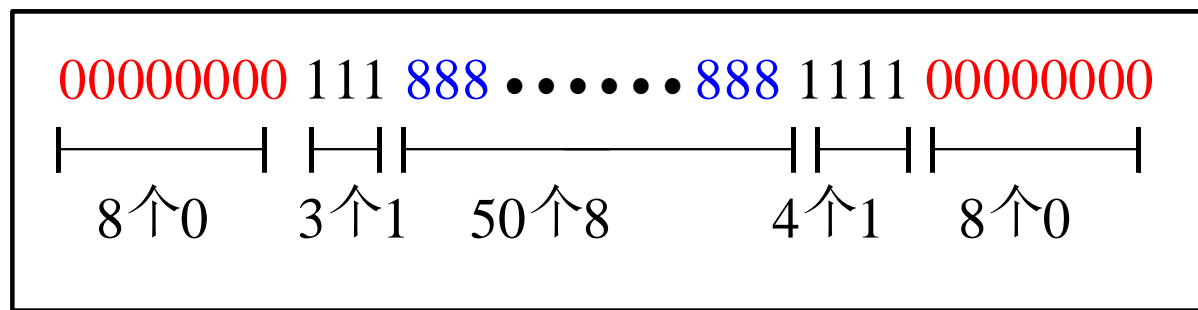
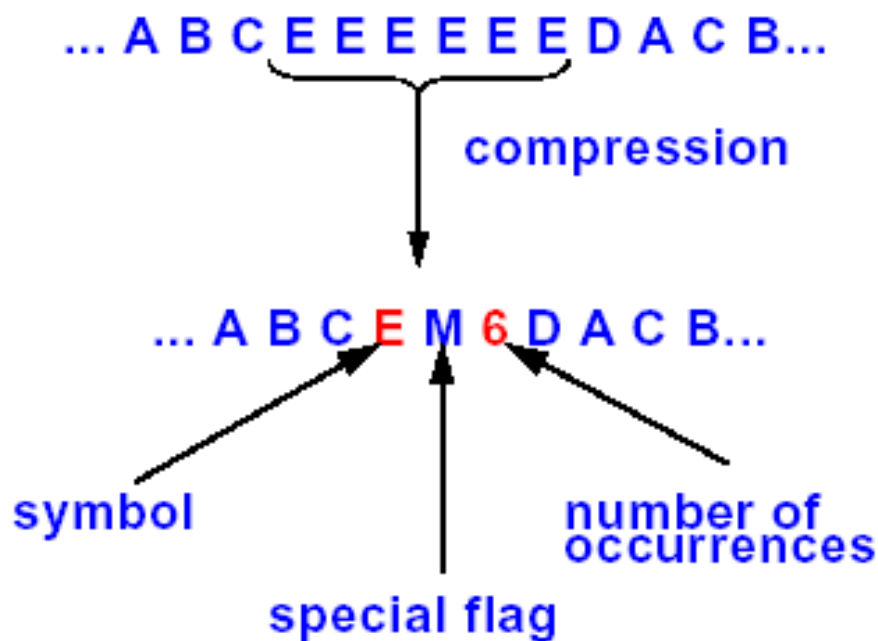


图3-2 RLE编码概念

3.3 RLE编码(续)

◆ Assumption:

- Long sequences of identical symbols.
- Example,



3.4 统计编码——算术编码

◆ 算术编码(Arithmetic Coding)

- ✓ 给已知统计信息的符号分配代码的数据无损压缩技术
- ✓ 基本思想是用0和1之间的一个数值范围表示输入流中的一个字符，而不是给输入流中的每个字符分别指定一个码字
- ✓ 把整个信源表示为0到1之间的一个实数区间，其长度等于该序列的概率，再在该区间内选择一个代表性的小数，转化为二进制作作为实际的编码输出
- ✓ 实质上是为整个输入字符流分配一个“码字”，因此它的编码效率可接近于熵

3.4 算术编码举例

◆ [例1]

- ✓ 假设信源符号为{00, 01, 10, 11}, 它们的概率分别为{ 0.1, 0.4, 0.2, 0.3 }
- ✓ 对二进制消息序列10 00 11 00 10 11 01进行算术编码

3.4 算术编码举例(续1)

■ 初始化

- 根据信源符号的概率把间隔 $[0, 1)$ 分成如表3-4所示的4个子间隔： $[0, 0.1)$, $[0.1, 0.5)$, $[0.5, 0.7)$, $[0.7, 1)$ 。其中 $[x, y)$ 的表示半开放间隔，即包含 x 不包含 y ， x 称为低边界或左边界， y 称为高边界或右边界

表3-4 例1的信源符号概率和初始编码间隔

符号	00	01	10	11
概率	0.1	0.4	0.2	0.3
初始编码间隔	$[0, 0.1)$	$[0.1, 0.5)$	$[0.5, 0.7)$	$[0.7, 1]$

3.4 算术编码举例(续2)

- ◆ 确定符号的编码范围

$$\text{low}_{i+1} = \text{low}_i + \text{range}_i * \text{range_low}$$

$$\text{high}_{i+1} = \text{low}_i + \text{range}_i * \text{range_high}$$

- ◆ 整个编码过程如图3-3所示

- ◆ 编码和译码的全过程分别见表3-5和表3-6

3.4 算术编码举例(续3)

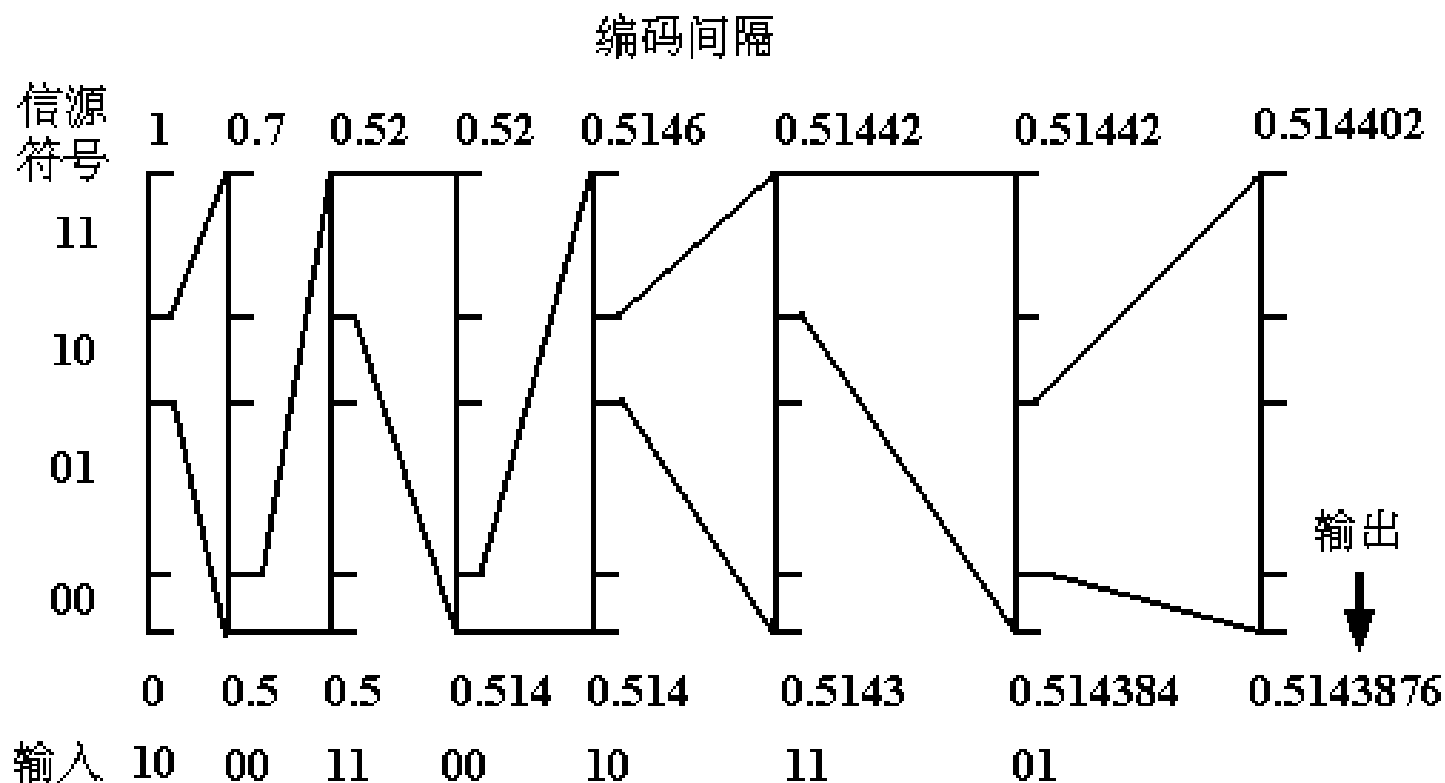


图3-3 算术编码过程

3.4 算术编码举例(续4)

表 3-5 编码过程

步骤	输入符号	编码间隔	编码判决
1	10	[0.5, 0.7]	符号的间隔范围[0.5, 0.7)
2	00	[0.5, 0.52]	[0.5, 0.7]间隔的第一个 1/10
3	11	[0.514, 0.52]	[0.5, 0.52]间隔的最后三个 1/10
4	00	[0.514, 0.5146]	[0.514, 0.52]间隔的第一个 1/10
5	10	[0.5143, 0.51442]	[0.514, 0.5146]间隔的第五个 1/10 开始, 2 个 1/10
6	11	[0.514384, 0.51442]	[0.5143, 0.51442]间隔的最后三个 1/10
7	01	[0.5143876, 0.514402]	[0.514384, 0.51442]间隔的 4 个 1/10, 从第一个 1/10 开始
8	从[0.5143876, 0.514402]中选择 一个 个数(如 0.5143876)作为输出		

表 3-6 译码过程

步骤	间隔	译码符号	译码判决
1	[0.5, 0.7]	10	0.51439 在间隔 [0.5, 0.7)
2	[0.5, 0.52]	00	0.51439 在间隔 [0.5, 0.7)的第 1 个 1/10
3	[0.514, 0.52]	11	0.51439 在间隔[0.5, 0.52)的第 7 个 1/10
4	[0.514, 0.5146]	00	0.51439 在间隔[0.514, 0.52)的第 1 个 1/10
5	[0.5143, 0.51442]	10	0.51439 在间隔[0.514, 0.5146)的第 5 个 1/10
6	[0.514384, 0.51442]	11	0.51439 在间隔[0.5143, 0.51442)的第 7 个 1/10
7	[0.51439, 0.5143948]	01	0.51439 在间隔[0.51439, 0.5143948]的第 1 个 1/10
7	译码的消息: 10 00 11 00 10 11 01		

3.4 算术编码举例(续5)

◆ 将区间中的小数转换为二进制数，生成码字

Procedure 7.2 (Generating Codeword for Encoder).

```
BEGIN
    code = 0;
    k = 1;
    while (value(code) < low)
    {
        assign 1 to the kth binary fraction bit;
        if (value(code) > high)
            replace the kth bit by 0;
        k = k + 1;
    }
END
```

3.4 算术编码特性

- ◆ 由于实际的计算机精度不可能无限长，运算中会出现溢出问题。
- ◆ 算术编码器对整个信源只产生一个码字，为 $[0,1)$ 之间的一个实数，因此译码须在接收到这个实数后才能译码。
- ◆ 算术编码是一种对错误很敏感的方法。

3.5 词典编码(Dictionary Coding)

- ◆ 词典编码是文本中的词用它在词典中表示位置的号码代替的一种无损数据压缩方法。
- ◆ 词典编码根据数据本身包含有重复编码这个特性
- ◆ 词典是指（1）用以前处理过的数据来表示编码过程中遇到的重复部分；（2）可以是任意字符的组合，编码数据过程中当遇到已经在词典中出现的“短语”时，编码器就输出这个词典中的短语的“索引号”，而不是短语本身。

3.5 词典编码

- 采用静态词典编码技术时，编码器需要事先构造词典，解码器要事先知道词典。
- 采用动态词典编码技术时，编码器将从被压缩的文本中自动导出词典，解码器解码时边解码边构造解码词典
- 代表算法：LZ77算法，LZSS算法，LZ78算法，LZW算法

3.5 词典编码

◆ 第一类编码算法

- 用已经出现过的字符串替代重复的部分
- 编码器的输出仅仅是指向早期出现过的字符串的“指针”

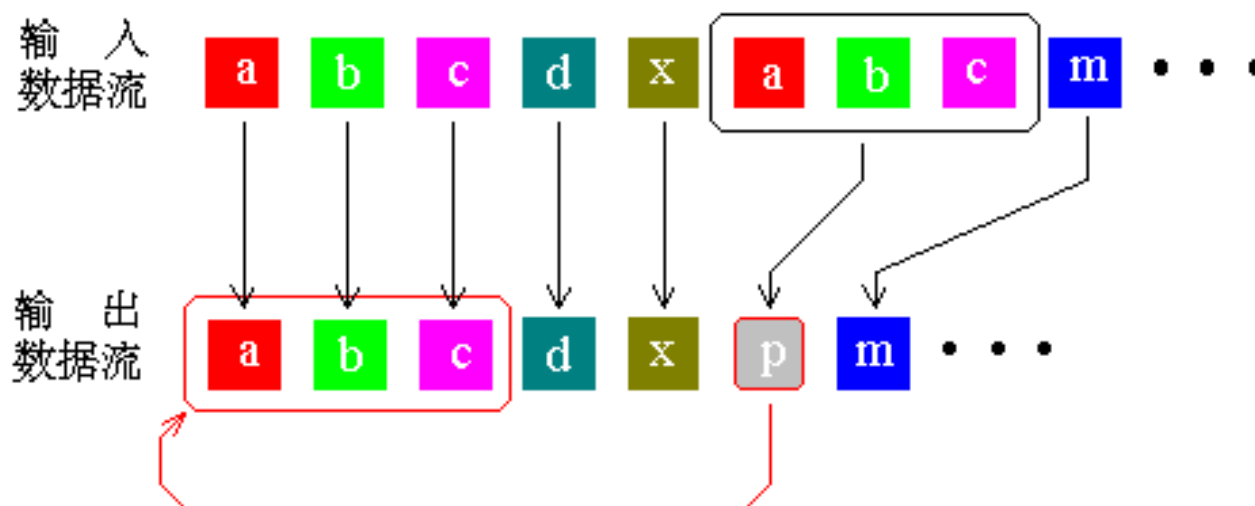


图3-4 第一类词典编码概念

3.5 词典编码

◆ 第二类编码算法

- 从输入的数据中创建一个“短语词典(dictionary of the phrases)”
- 编码器输出词典中的短语“索引号”，而不是短语

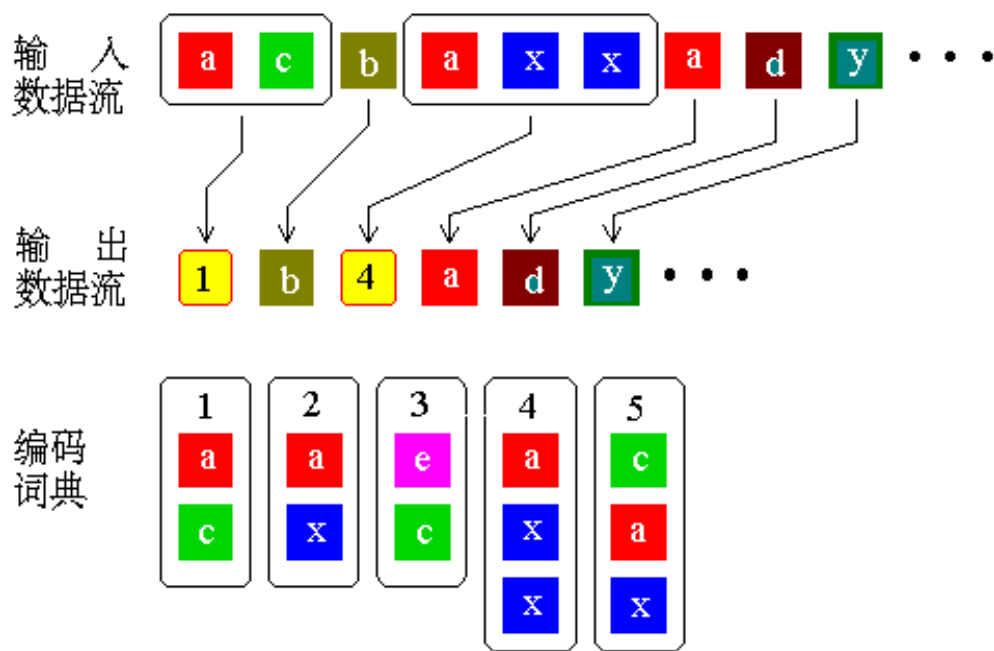


图3-5 第二类词典编码概念

3.5 词典编码

◆ LZW(Lempel-Ziv-Welch)简介

- 由Abraham Lempel、Jacob Ziv与Terry Welch提出的一种无损数据压缩算法，简称 LZW 的压缩算法，可以用任何一种语言来实现它。
- LZW算法是一种通用的压缩方法，可用于任何二值数据文件的压缩，它也是目前最为流行的一种数字图像压缩方法。
- LZW是GIF图像文件的压缩算法，而且zip压缩的思想也是基于LZW实现的，所以LZW对文本文件具有很好的压缩性能。也成功地应用于高速硬盘控制器上。
- LZW编码对信源符号的可变长度序列分配固定长度的码字，且不需要了解被编码信源的概率情况。

3.5 词典编码

◆ LZW算法基本原理

- 提取原始文本文件数据中的不同字符，基于这些字符创建一个编译表，然后用编译表中的字符的索引来替代原始文本文件数据中的相应字符，减少原始数据大小。这里的编译表不是事先创建好的，而是根据原始文件数据动态创建的，解码时还要从已编码的数据中还原出原来的编译表。
- 图像可以当作一个一维的比特串，算法在产生输出串的同时更新编码表。

3.5 词典编码

◆ LZW算法基本步骤

1. 将词典初始化，使其所包含所有可能的单字符。当前前缀P初始化为空。
2. 当前字符C的内容为输入字符流中的下一个字符。
3. 判断P+C是否在词典中：如果是，则用C扩展P，即让 $P=P+C$ 。如果否，则①输出当前前缀P的码字到码字流；②将P+C添加到词典中；③令前缀 $P=C$ （即现在的P仅包含一个字符C）。
4. 判断输入字符流中是否还有码字要编码：如果是，则返回到步骤2.如果否，则①把当前前缀P的码字输出到码字流；②结束。

3.5 词典编码

- ◆ 例： 对一个最简单的三字符A,B,C组成的字符串ABBABABAC进行LZW编码.

表1 待编码的字符串

位置	1	2	3	4	5	6	7	8	9
字符	A	B	B	A	B	A	B	A	C

3.5 词典编码

表2 LZW的编码过程

步骤	词典		输出
	(1)	A	
	(2)	B	
	(3)	C	
1	(4)	AB	(1)
2	(5)	BB	(2)
3	(6)	BA	(2)
4	(7)	ABA	(4)
5	(8)	ABAC	(7)
6	--	--	(3)

3.5 词典编码

表3 LZW的译码过程

步骤	词典		输入代码	输出
	(1)	A		
	(2)	B		
	(3)	C		
1	--	--	(1)	A
2	(4)	AB	(2)	B
3	(5)	BB	(2)	B
4	(6)	BA	(4)	AB
5	(7)	ABA	(7)	ABA
6	(8)	ABAC	(3)	C

3.5 词典编码

- ◆ 假设每个字符和编码都是一个字节
压缩比: $9/6=1.5$
- ◆ 典型的文本数据LZW编码使用12位的码长
压缩比: $(9*8) / (6*12) = 1$
- ◆ 一般而言, 当文本长度达到百万字节时, LZW压缩比的效果比较明显。

3.5 词典编码

◆ LZW Demo

https://www2.cs.sfu.ca/mmmbook/demos/lzw_encoding/

Pseudo Code

```
s = next input character;
while not EOF
{
    c = next input character;
    if s + c exists in the dictionary
        s = s + c;
    else
        output the code for s;
        add string s + c to the dictionary with a new code;
        s = c;
}
output the code for s;
```


3.5 词典编码

◆ LZW解压缩（简化版）

存在的问题：找不到输入编码(code)对应的字典条目(string)

Algorithm 7.3 (LZW Decompression (Simple Version)).

```
BEGIN
  s = NIL;
  while not EOF
  {
    k = next input code;
    entry = dictionary entry for k;
    output entry;
    if (s != NIL)
      add string s + entry[0] to dictionary
      with a new code;
    s = entry;
  }
END
```



3.5 词典编码

◆ LZW解压缩 (改进版)

异常：找不到某个编码(code)对应的字符串(string)

处理：能够通过检查解码器端的字典中是否定义了输入的编码(code)来处理这种异常

Algorithm 7.4 (LZW Decompression (Modified)).

BEGIN

 s = NIL;

 while not EOF

 {

 k = next input code;

 entry = dictionary entry for k;

 /* exception handler */

 if (entry == NULL)

 entry = s + s[0];

 output entry;

 if (s != NIL)

 add string s + entry[0] to dictionary
 with a new code;

 s = entry;

 }

END
