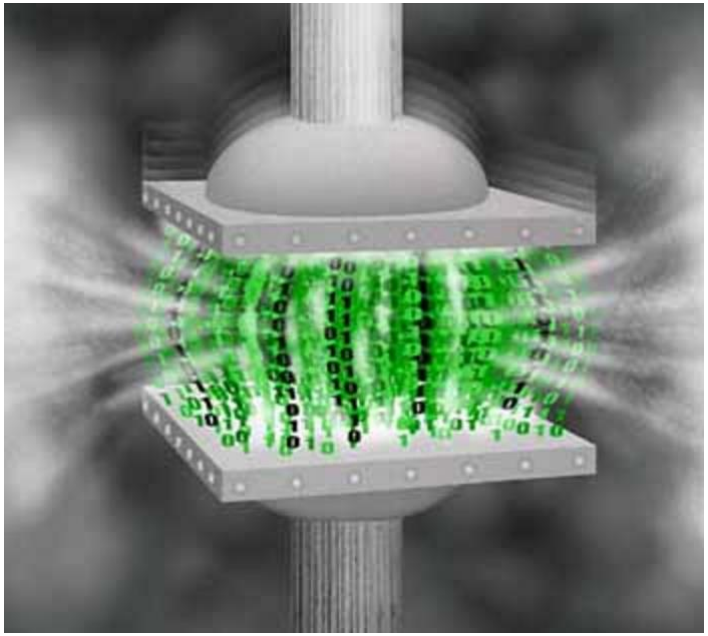
第2章 数据无损压缩



目录

2.0 简介

2.1 信息论基础知识

信息量、熵、信源编码

决策量、编码冗余

2.2 统计编码

2.2.1 香农-范诺编码

2.2.2 霍夫曼编码

2.2.3 算术编码

2.3 行程长度编码

2.4 词典编码

2.4.1 词典编码的思想

2.4.2 LZ77算法

2.4.3 LZSS算法

2.4.4 LZ78算法

2.4.5 LZW算法

目录

2.0 简介

2.1 信息论基础知识

信息量、熵、信源编码

决策量、编码冗余

2.2 统计编码

2.2.1 香农-范诺编码

2.2.2 霍夫曼编码

2.2.3 算术编码

2.3 行程长度编码

2.4 词典编码

2.4.1 词典编码的思想

2.4.2 LZ77算法

2.4.3 LZSS算法

2.4.4 LZ78算法

2.4.5 LZW算法

2.0 为什么要压缩？

未经压缩的高清电视（HDTV）数据率：

1280×720 color pixels ×24 bpp ×60 frames/s ＝ 1.3Gbps

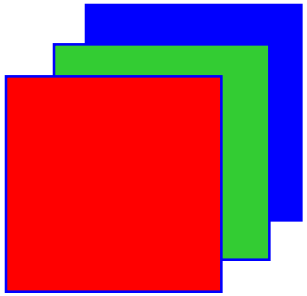
一张50GB的蓝光光碟能存储上述视频的时长:

50GB/1.3Gbps = 307.7seconds

要在带宽为19.3Mbits/s的HDTV广播频道上传输该视频，最小的压缩比为：

1.3Gbps/19.3Mbps=67.4

2.0 为什么要压缩？



计算题：

1.**图像：** 一张640×480真彩(24位)的原始图像需要多大的存储空间？

640×480×24＝7372800(bit)=900KB 其中 (1Byte=8bit)

2.**视频：** 这样的图像构成视频，以每秒30帧进行播放，所需传输率为多少？

7372800*30=221184000(b/s)≈221Mbps

3.**存储：** 一张650MB(5200Mb)的CD光盘能存储上述视频的时长为多少？

650MB/221Mbps = 23.5seconds

2.0 多媒体数据压缩与编码

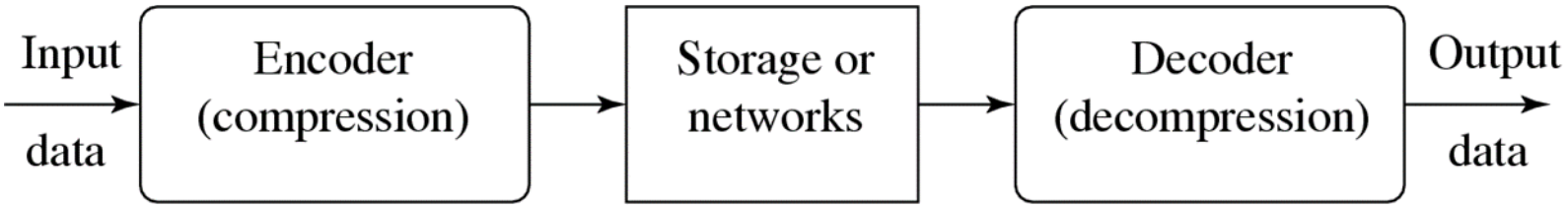
- 为什么要压缩？
  - 多媒体数据的数据量大
  - 降低多媒体数据对存储器容量的要求
  - 降低多媒体数据对传输带宽的要求

2.0 什么是压缩？

- 压缩：

  减少表达一个数字信号（例如声音、图像、视频、或其他信号）所用比特数，同时

  - 能够完全重建出原始数据（无损压缩）或
  - 恢复出质量可被接受的原始数据（有损压缩）



2.0 压缩示例

(a)

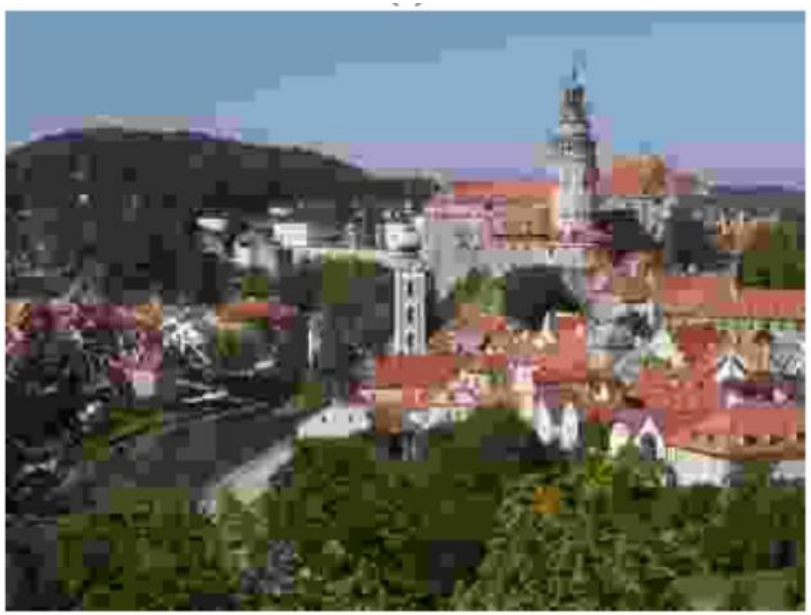
原始图像

(b)

质量因子Q=75时的JPEG解压缩图像，文件压缩至原始大小的7.11%

2.0 压缩示例

(c)



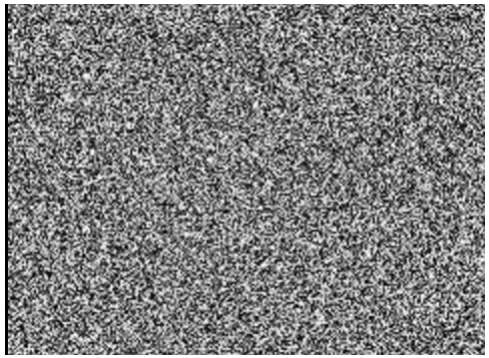
(d)

Q=25时的JPEG解压缩图像，文件压缩至原始大小的3.13%

Q=5时的JPEG解压缩图像，文件压缩至原始大小的1.13%

2.0 压缩的依据

- 数据冗余：统计冗余或数据中出现的结构（空域内、时域内、频域内）



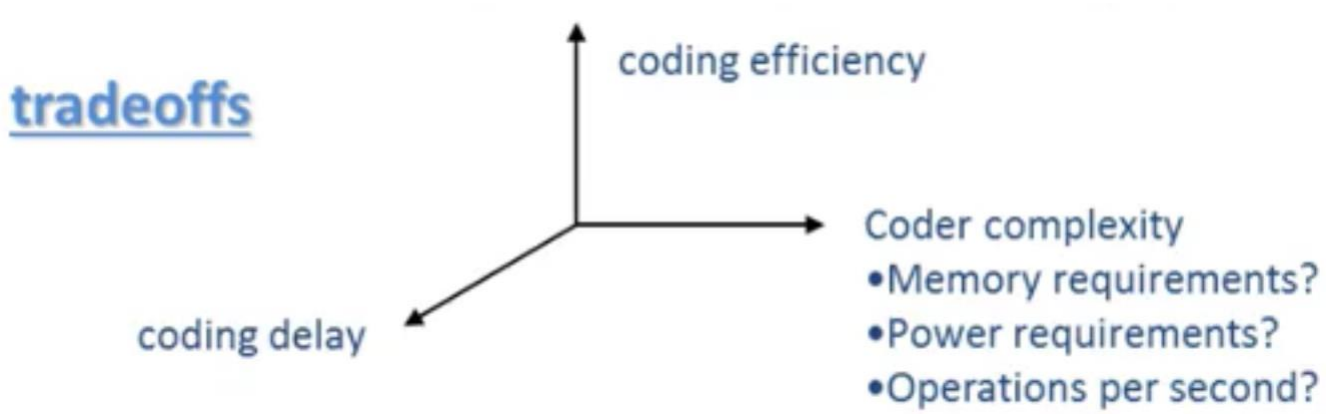
- 视听冗余：感知上多余的信息
  - 听觉系统的敏感度有限
  - 视觉系统的敏感度有限

2.0 多媒体与压缩

- 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)/视频(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)

2.0 无损压缩

- 目标：由压缩数据能精确重建原始数据，即过程可逆。
- 压缩比由信源的熵来决定的
  - 无损压缩的压缩比相对较低
- 应用举例：文字、医学图像、有损编码器



2.0 无损信源压缩

- 数据无损压缩的方法
  - 香农范诺编码(Shannon-Fano Coding)
  - 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)

目录

2.0 简介

2.1 信息论基础知识

决策量、信息量、熵、信源编码、编码冗余

2.2 统计编码

2.2.1 香农-范诺编码

2.2.2 霍夫曼编码

2.2.3 算术编码

2.3 行程长度编码

2.4 词典编码

2.4.1 词典编码的思想

2.4.2 LZ77算法

2.4.3 LZSS算法

2.4.4 LZ78算法

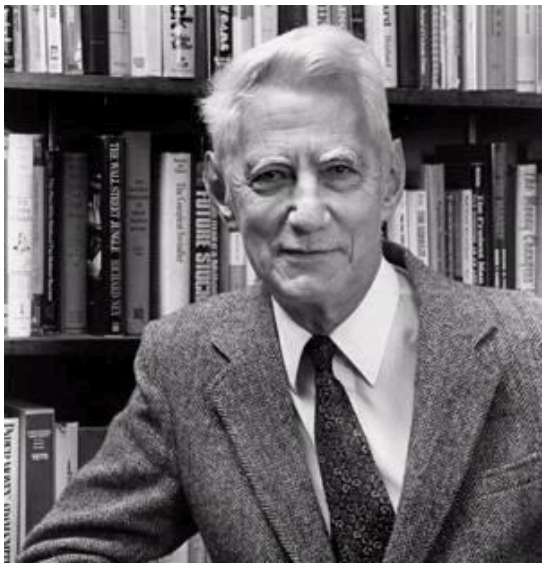
2.4.5 LZW算法

2.1 信息论基础知识

- 数据无损压缩的理论——信息论(information theory)
  - 1948年创建的数学理论的一个分支学科，研究信息的编码、传输和存储
  - 最初应用于通信工程领域，后扩展到包括计算在内的多个领域，如信息的存储、信息的检索等。

2.1 信息论基础知识

- 信息论之父 Claude Elwood Shannon(1916-2001)



- In 1948:
  - 发表了“*A mathematical theory of communication*”，提议用二进制数据对信息进行编码
  - Shannon理论的一个重要贡献: 熵的概念

2.1 信息论基础知识

- 信息产生的过程可以看成一个信源发出一串随机得从有限符号集里挑选的符号
  - 例如：手写英文；n-bit图像

- 最简单情况：离散无记忆信源（discrete memoryless source, DMS)
  - 信源相继产生的符号在统计上是独立的,且服从同一分布

2.1 熵(Entropy)

- 自信息量(self information)：

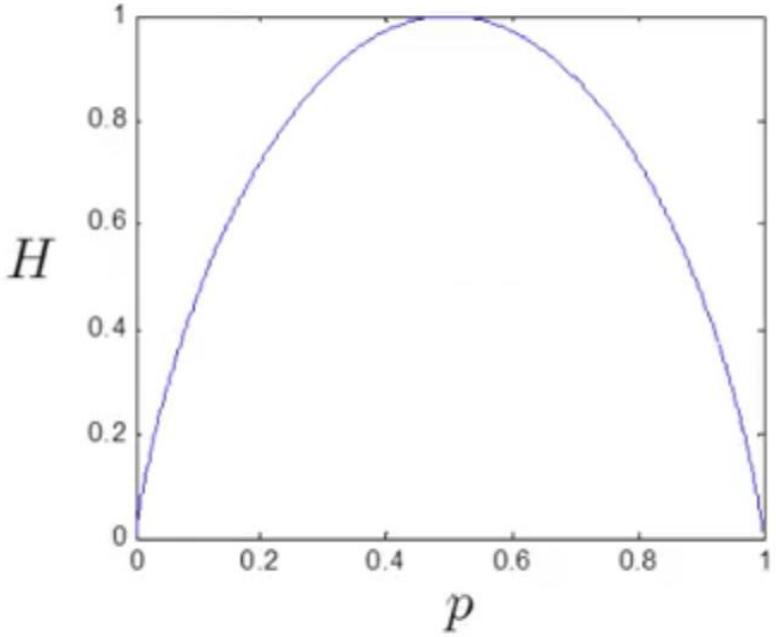
$$I(S_i) = log\frac{1}{p_i} = -logp_i$$

  - 一个概率较小的事件发生能提供更多的信息量
  - 多个独立事件作为一个事件产生的信息量相当于各个事件自信息量的总和
- DMS的熵（entropy)：信源符号的信息量的平均值

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

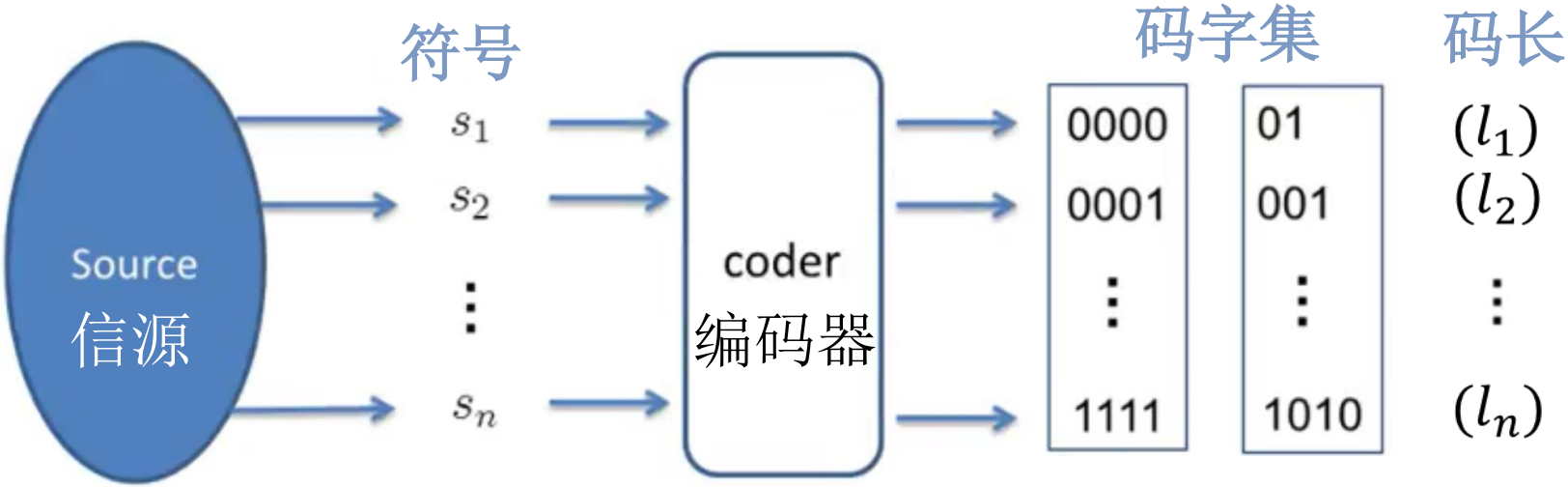
2.1 熵-示例

- 离散无记忆信源的熵



$$H = -plog_2 p - (1-p)\log_2(1-p)$$

2.1 编码



符号集 $A = \{s_1, s_2, \ldots, s_n\}$

p_1 p_2 p_n

平均码字长度: $l_{avg} = \sum_{i=1}^{n} l_i p_i$

2.1 信源编码定理

- S为一个符号集大小为n、熵为H(S)的信源。考虑把N个信源符号编成二进制的码字。对于任意的δ>0，都可以通过选择足够大的N来构建码字，使得原始符号的平均比特数l_{avg}满足：

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

2.1 信源的熵

- 英语的熵
  - DMS, 相同概率： H(S) = 4.70bits/letter
  - DMS, 实际概率： H(S) = 4.14 bits/letter

- 利用数据的结构
  - 例子1：考虑数据：1234323456
  - 例子2：考虑数据：334533334545

2.1 编码示例

符号	概率
s_1	1/2
s_2	1/4
s_3	1/8
s_4	1/8
平均码长	

熵H(S)=1.75 bits/符号

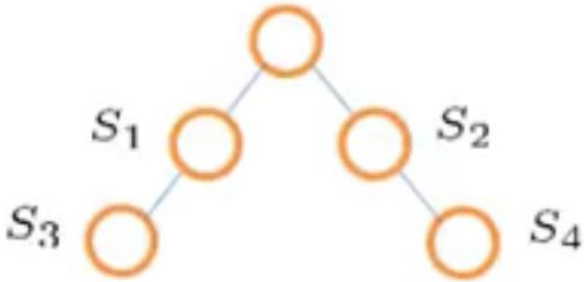
- 可唯一解码：任何一段有限码字序列对应唯一的信息序列
- 前缀码字：每个码字都不是其他码字的前缀

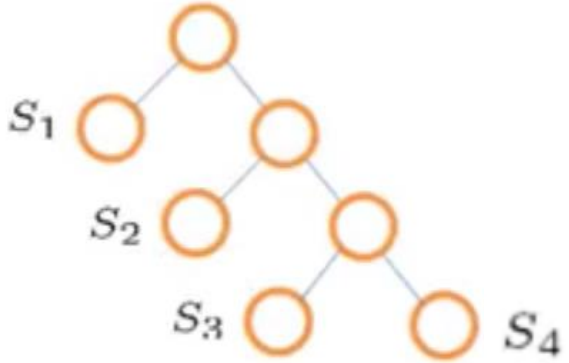
2.1 码字的二叉树表示

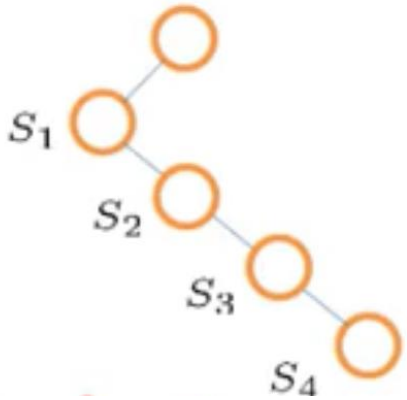
Code 2
0
1
00
11

Code 3
0
10
110
111

Code 4
0
01
011
0111







2.1 信息论基础知识

- 重要概念
  - 压缩比：

    压缩比 = 压缩前文件大小/压缩后文件大小
  - 熵：

$$H(S) = -\sum_i p_i \log(p_i)$$
  - 平均码长：

$$l_{avg} = \sum_i p_i l_i$$
  - 冗余：

$$\rho = l_{avg} - H(s)$$

目录

2.0 简介

2.1 信息论基础知识

信息量、熵、信源编码、决策量、编码冗余

2.2 统计编码

2.2.1 香农-范诺编码

2.2.2 霍夫曼编码

2.2.3 算术编码

2.3 行程长度编码

2.4 词典编码

2.4.1 词典编码的思想

2.4.2 LZ77算法

2.4.3 LZSS算法

2.4.4 LZ78算法

2.4.5 LZW算法

2.2 统计编码

- 统计编码
  - 给已知统计信息的符号分配代码的无损压缩方法
- 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原理相同，都根据符号集中各符号出现的频繁程度来编码
  - 算术编码使用0和1间的实数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

2.2.1香农-范诺(Shannon-Fano)编码

- 最早阐述和实现这种编码的是Shannon(1948年)和Fano(1949年)，采用从上到下的方法进行编码。

- Shannon-Fano编码步骤：

  （1）按照符号出现的频度或概率排序

  （2）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，直至所有部分只含有一个符号

2.2.1 香农-范诺编码(例1）

- 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，如表格所示

    (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) 计算该图像可能获得的压缩比的实际值

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	15/40	7/40	7/40	6/40	5/40

2.2.1 香农-范诺编码(例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

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

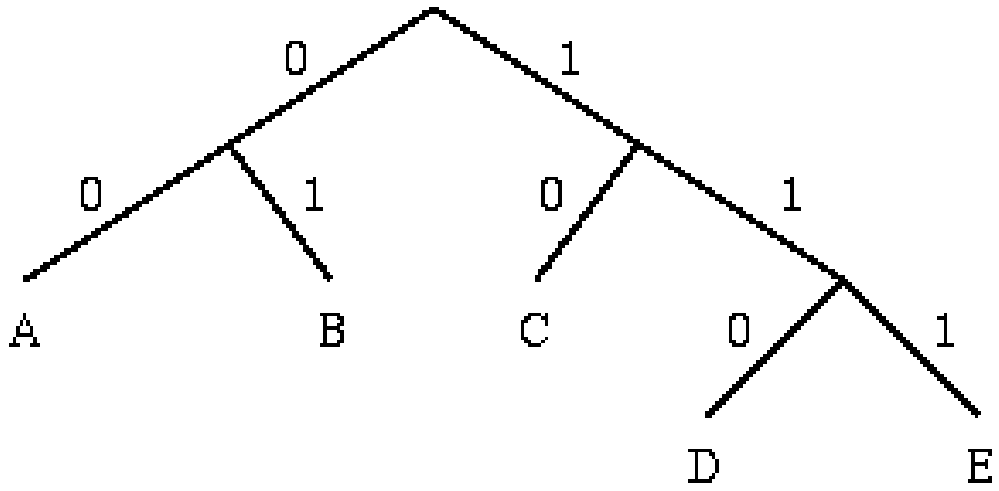
$$= -p(A)\log_2(p(A)) - p(B)\log_2(p(B)) - \cdots - p(E)\log_2(p(E))$$

$$= (15/40)\log_2(40/15) + (7/40)\log_2(40/7) + \ldots + (5/40)\log_2(40/5) \approx 2.196\text{bits}$$

  - 40个像素只需用87.84位
    理论压缩比为120:87.84≈1.37:1
    即3:2.196≈1.37

2.2.1香农-范诺编码(例1）

- （2）Shannon-Fano编码



香农-范诺算法编码举例

2.2.1香农-范诺编码(例1）

符号	出现的次数(P_i)	$\log_2(1/P_i)$	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）压缩比的实际值

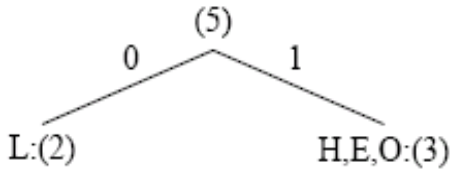
- 按照这种方法进行编码需要的总位数为30+14+14+18+15＝91，实际的压缩比为120:91≈1.32 : 1

2.2.1香农-范诺编码（例2）

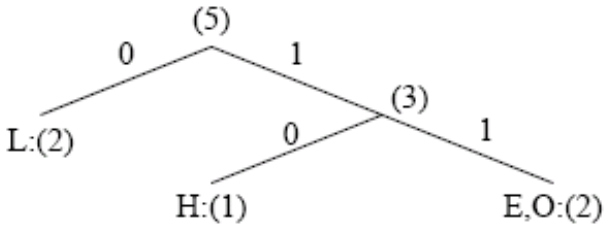
- 香农范诺算法

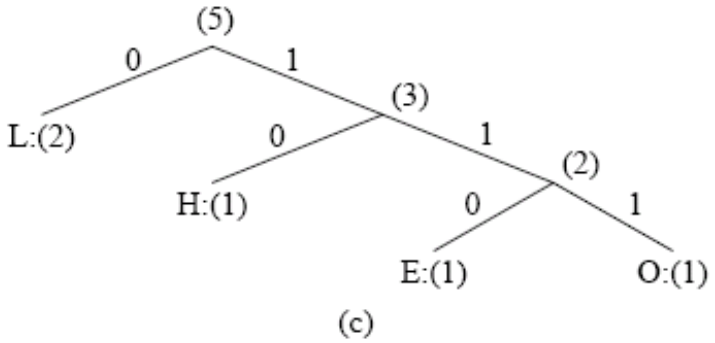
假设对单词HELLO中字符进行编码，每个字符出现的频率如右：

符号	H E L O
数目	1 1 2 1



(a)





(c)

Symbol	Count	$\log_2 \frac{1}{p_i}$	Code	# of bits used
L	2	1.32	0	2
H	1	2.32	10	2
E	1	2.32	110	3
O	1	2.32	111	3
TOTAL number of bits:				10

2.2.2 霍夫曼(Huffman）编码

- 出现概率大的信源符号赋于短码字
- 两个最少出现的符号会有相同长度的码字
- Huffman编码步骤：

  （1）按照概率对符号进行排序；

  （2）概率最小的两个符号组成一个新符号，概率等于对应符号的概率和（进行这一步，符号集大小减1）；

  （3）对剩下的符号重复这个过程，直至只剩下一个符号；

  （4）从根节点顺着树枝到叶子，从编码过程对应的二叉树上得到码字。

2.2.2 霍夫曼编码（例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：**BABACACADADABBCBABEBEDDABEEEBB**
  - 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

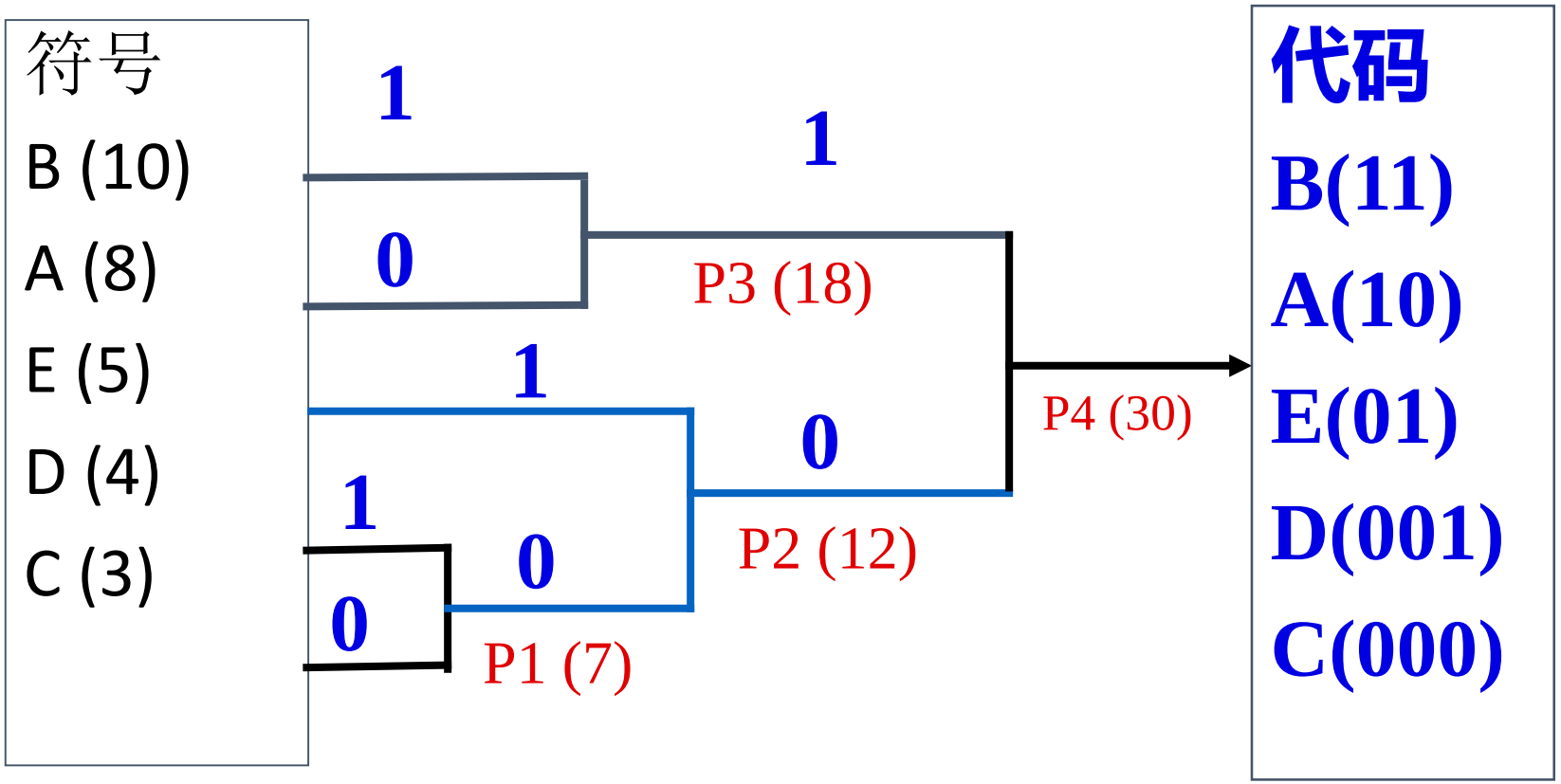
    (4) 编码前后的压缩比

2.2.2 霍夫曼编码（例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			

2.2.2 霍夫曼编码（例1）


符号
B (10)
A (8)
E (5)
D (4)
C (3)
1
0
1
0
1
0
1
0
P3 (18)
P2 (12)
P1 (7)
P4 (30)
代码
B(11)
A(10)
E(01)
D(001)
C(000)

2.2.2 霍夫曼编码（例1）

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	010	9
D	4	2.907	011	12
E	5	2.585	00	10
合计	30	1.0		67

30个字符组成的字符串需要67位

2.2.2 霍夫曼编码（例1）

(2) 计算该字符串的熵

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

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

并满足 $\sum_{i=1}^{n}p(x_i)=1$

- $H(S)$

$$\begin{aligned}
&=(8/30)\times\log_2(30/8)+(10/30)\times\log_2(30/10)+\\
&\quad(3/30)\times\log_2(30/3)+(4/30)\times\log_2(30/4)+\\
&\quad(5/30)\times\log_2(30/5)\\
&=[30\lg30-(8\times\lg8+10\times\lg10+3\times\lg3+4\\
&\quad\times\lg4+5\lg5)]/(30\times\log_2 2)\\
&=(44.3136-24.5592)/9.0308=2.1874\ \ (\text{Sh})
\end{aligned}$$

2.2.2 霍夫曼编码（例1）

(3) 计算该字符串的平均码长

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

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

$=2.233$ 位/符号

平均码长：67/30=2.233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

压缩比: 90/67=1.34:1

2.2.2 霍夫曼编码（例2）

- 霍夫曼编码举例

例：设有7个符号：　a_1, a_2, a_3, a_4, a_5, a_6, a_7

它们出现概率是：0.2, 0.19, 0.18, 0.17, 0.15, 0.1, 0.01

2.2.2 霍夫曼编码（分析）

每个编码均非其它码的前缀，因此唯一可译

(a_1=10,a_2= 11, a_3= 000, a_4= 001, a_5= 010, a_6= 0110, a_7=0111)

11 010 10 001 10 11 10 0111

a_2 a_5 a_1 a_4 a_1 a_2 a_1 a_7

1. 简单易实现，编码效率较高
2. 必须预先知道信源的统计特性

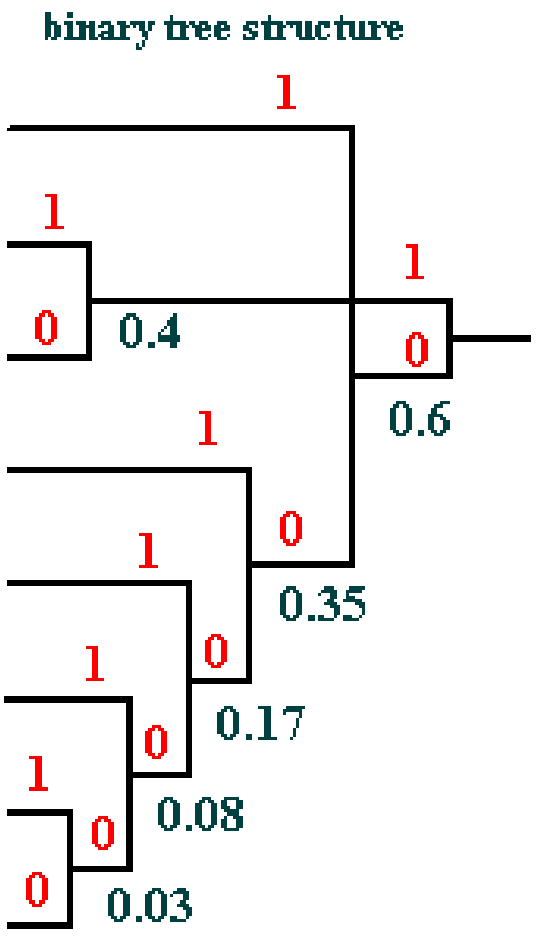
2.2.2 Huffman编码(例3)

- 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

- 对该字符串进行Huffman编码

2.2.2 Huffman编码(例3)

Original codewords	Huffman codewords	Symbol probabilities
111	01	P(h)=0.25
110	11	P(g)=0.2
101	10	P(f)=0.2
100	001	P(e)=0.18
011	0001	P(d)=0.09
010	00001	P(c)=0.05
001	000001	P(b)=0.02
000	000000	P(a)=0.01



2.2.2 Huffman编码（分析）

霍夫曼码是可变长度码，却不需要附加同步代码。

几个问题值得注意：

1. 没有错误保护功能；
2. 很难随意调用压缩文件中间内容进行译码；
3. 接收端需保存与发送端相同的霍夫曼码表。

Huffman编码-信源拓展

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

Letter	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

2.2.3 算术编码

- 算术编码(Arithmetic Coding)通常比Huffman编码效果更好
- Huffman编码为每个符号分配整数个比特的码字，而算术编码为整个输入符号流分配一个“码字”。
- 输入符号流用一个半开区间表示[a, b), 其中a和b是 0和1之间的实数。区间初始化为[0,1)。随着符号个数增加，区间长度越来越短，表示区间的比特数也越来越多。

2.2.3 算术编码-标签的生成

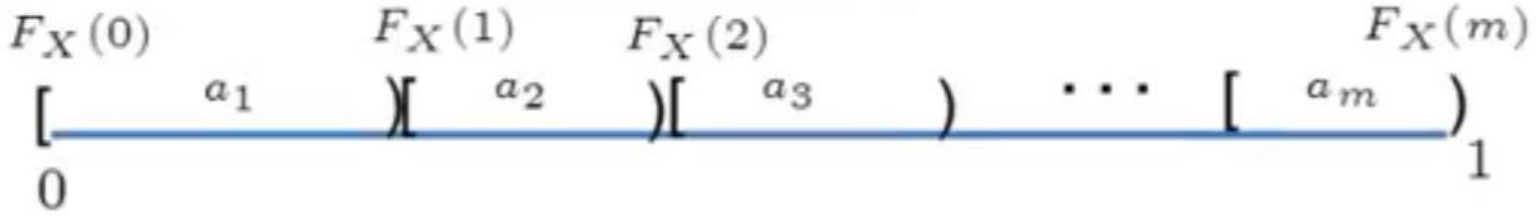
$\mathcal{A} = \{a_1, a_2, \cdots, a_m\}$; $P(a_i)$ Random variable: $X(a_i) = i$

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

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

- 核心思想：随着序列中收到的符号越来越多，标签所在的区间逐渐缩小。

把[0,1)单元区间划分成下面的子区间



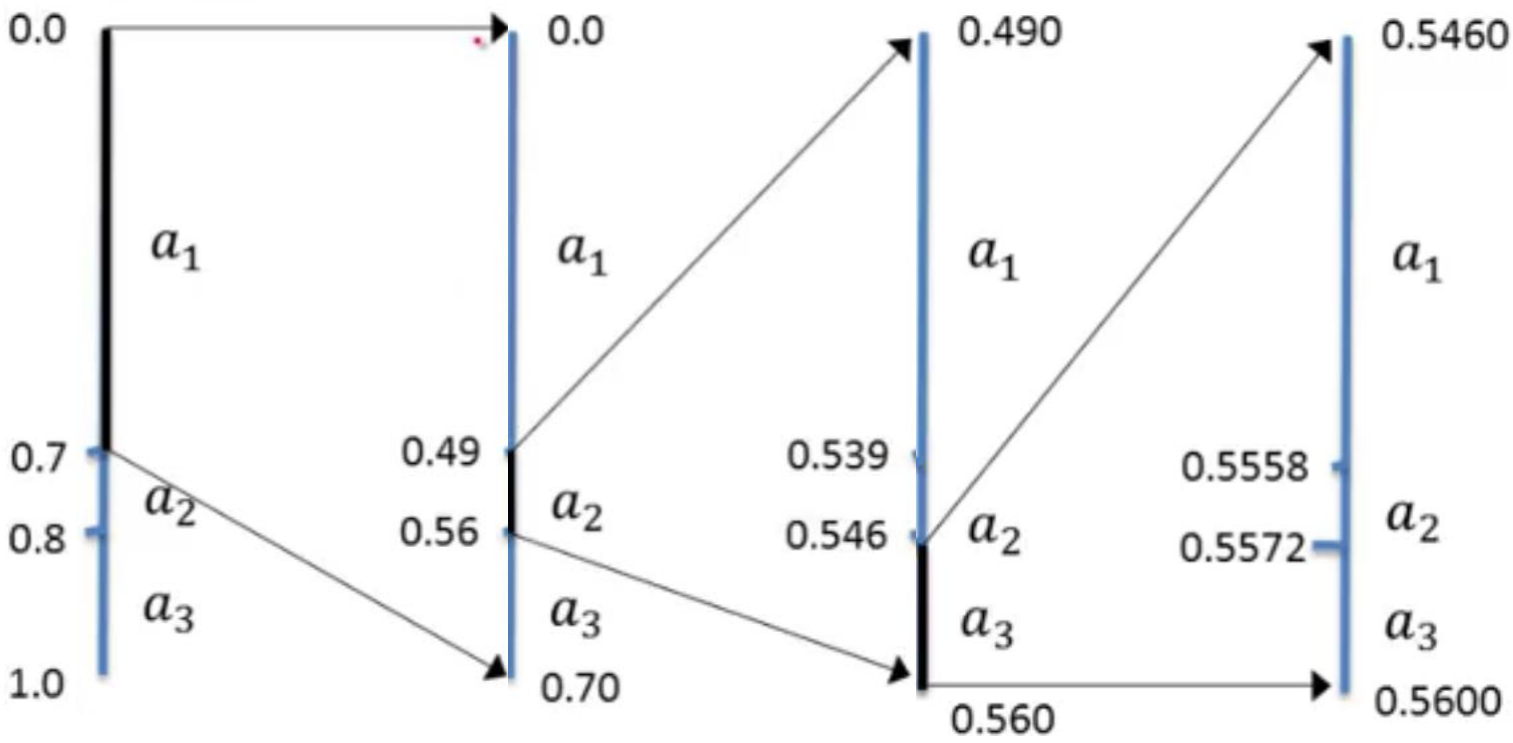
$a_i \in [F_x(i-1), F_X(i))$

2.2.3 算术编码算法描述

```
BEGIN
   low = 0.0;  high = 1.0;  range = 1.0;
   while (symbol != terminator)
   {
     get (symbol);
     low = low + range * Range_low(symbol);
     high = low + range * Range_high(symbol);
     range = high - low;
   }
   output a code so that low <= code < high;
END
```

2.2.3 算术编码-(例1）

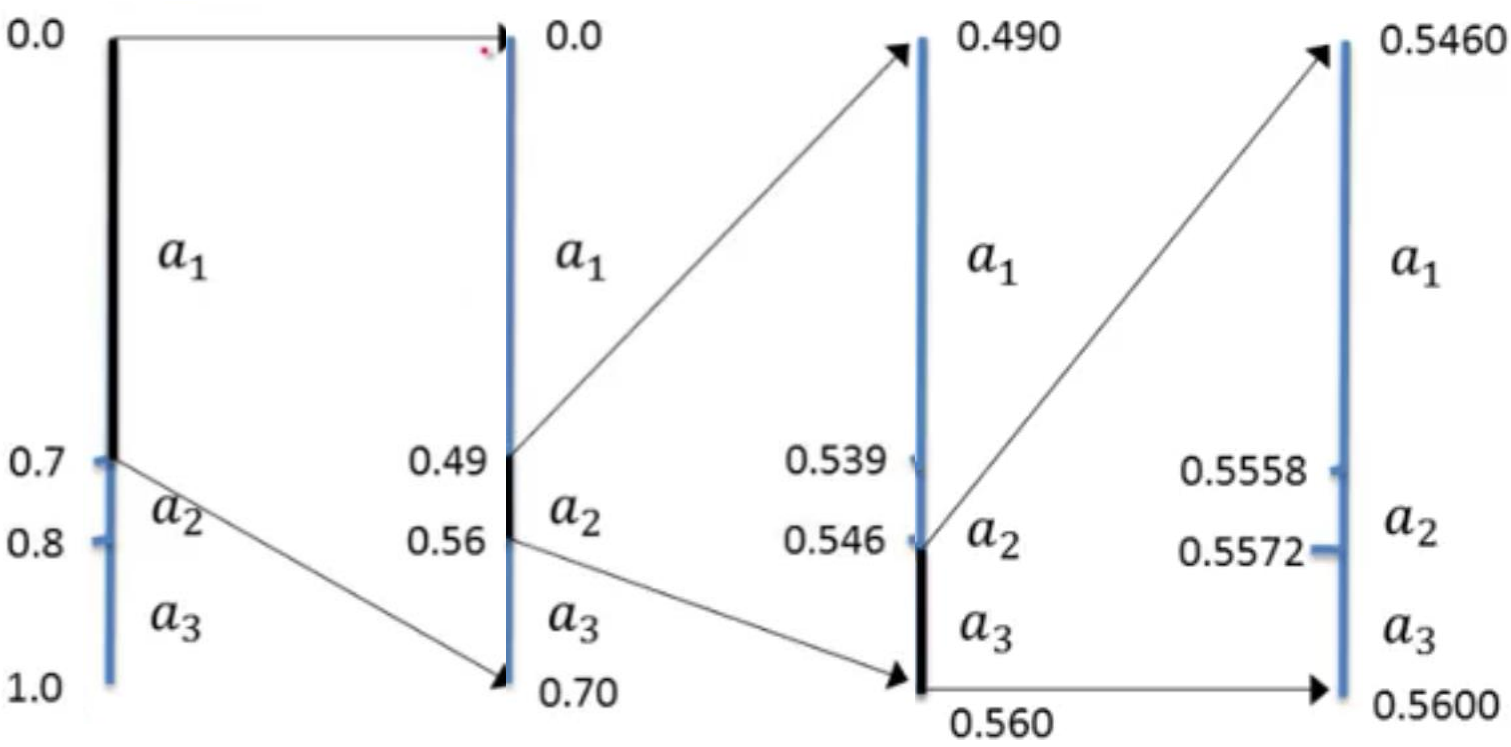
$\mathcal{A} = \{a_1, a_2, a_3\};\ P(a_1) = 0.7, P(a_2) = 0.1, P(a_3) = 0.2; F_X(1) = 0.7, F_X(2) = 0.8, F_X(3) = 1$



2.2.3 算术解码-(例1）

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

Tag=0.55 ， 3 symbols

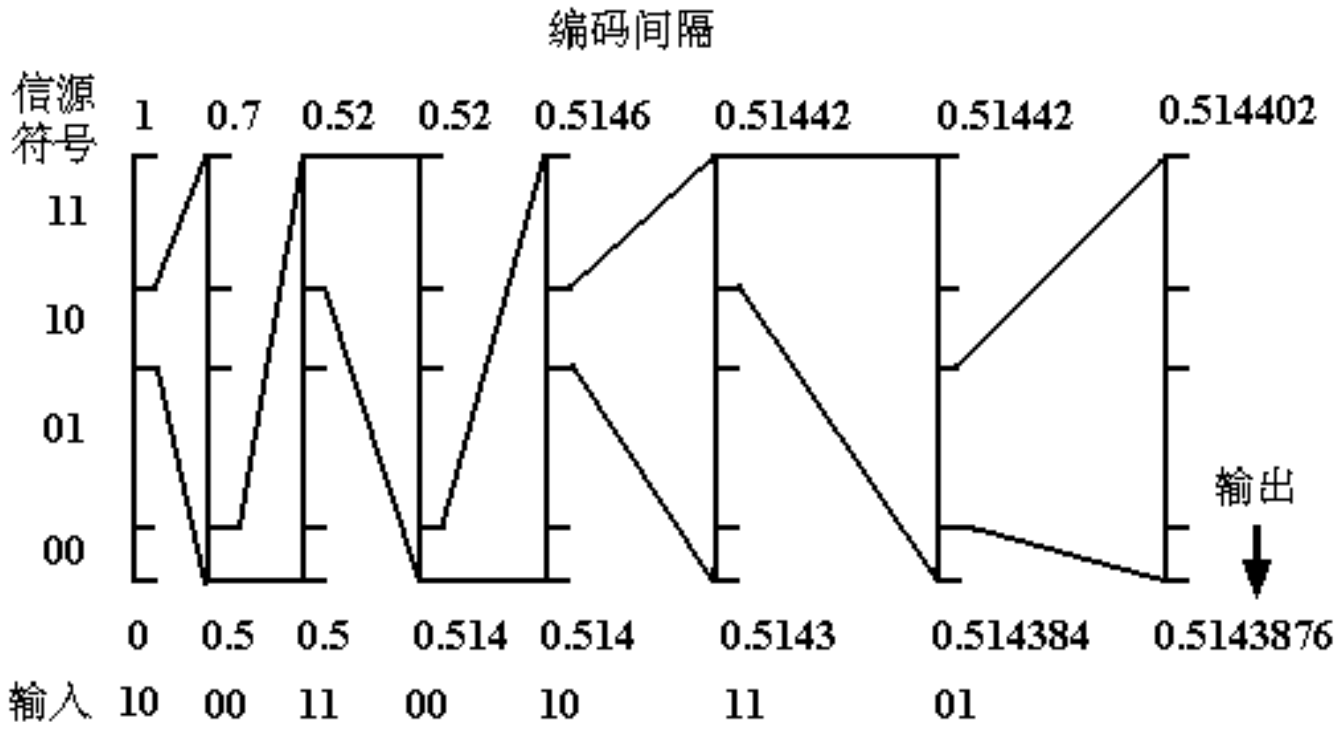


模仿编码过程

2.2.3 算术编码-(例2）

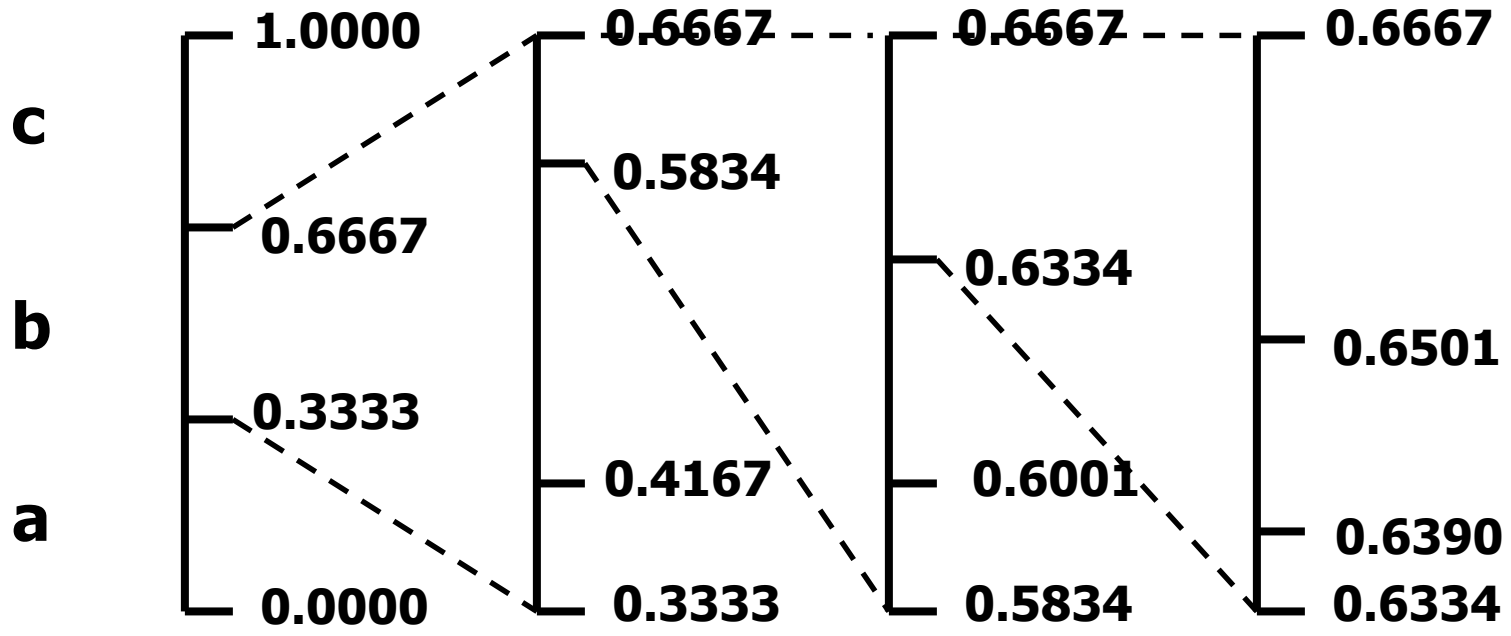
待编码的二进制消息序列的输入为：10 00 11 00 10 11 01

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)



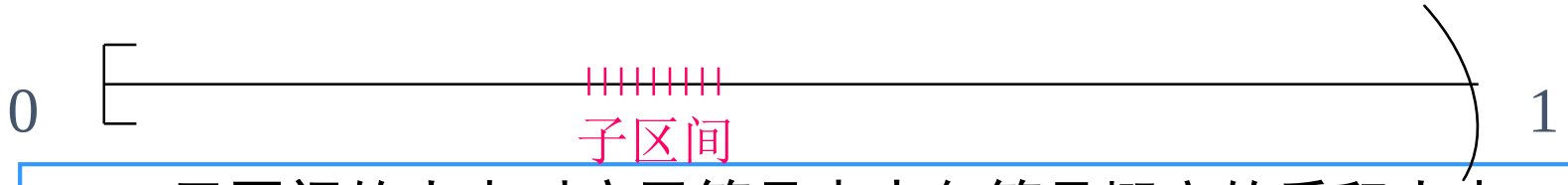
2.2.3 算术编码（补充：自适应算术编码）

输入序列为：bcc ....



c	1/3	1/4	2/5	3/6
b	1/3	2/4	2/5	2/6
a	1/3	1/4	1/5	1/6

2.2.3 算术编码（分析）



子区间的大小对应于符号串中各符号概率的乘积大小

子区间的位置对应着 符号串中符号的排列顺序

子区间的头尾均用二进制数表示

该符号串的编码是满足下列条件的、最短的数F：

头< = F < 尾

可见：

不同的串，对应着不同的子区间，选择子区间中任意的一个数作为该符号串的代码，则必有不同的编码

选择 子区间中最短的数作为 该符号串的代码

一般而言，子区间越宽，码长越短；子区间越窄，码长越长

2.2.3 算术编码生成码字

```
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
```

算术编码的最后一步为生成一个在区间[low,high)范围内的数。上述算法能保证找到最短的二进制码字。

2.2.3 算术编码（分析）

- 算术编码注意点

1. 由于实际计算机精度有限（16位、32位或者64位），运算中溢出是明显的问题，可用比例缩放法解决。

2. 算术编码器对消息序列只产生一个码字，这个码字是在[0, 1)中的一个实数，因此译码器在接受到表示这个实数的所有位之前不能进行译码。

3. 算术编码也是一种对错误很敏感的编码方法，如果有一位发生错误就会导致整个消息译错。

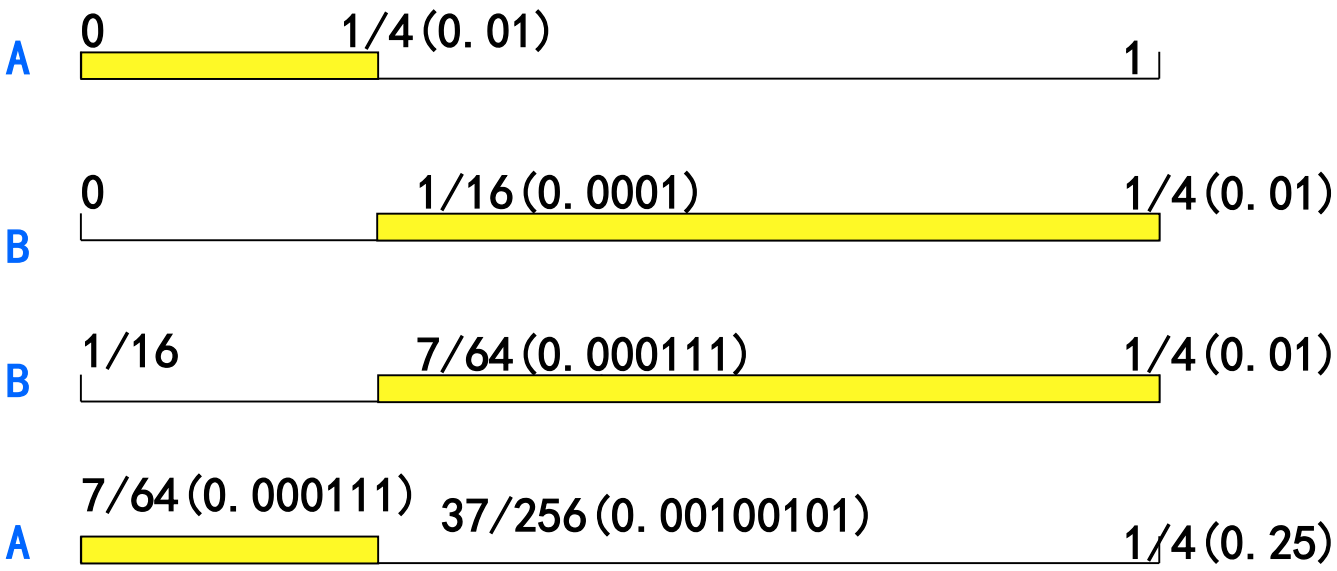
2.2.3 算术编码（思考）

信源符号及其概率如下:

符号	A	B
概率	0.25	0.75

求输入串ABBA的算术二进制编码(赋予大概率为右区间)，并求最短编码。

2.2.3 算术编码（答案）



ABBA符号串对应的区间为[7/64，37/256），即对应的二进制区间为[0.000111,0.00100101 ），取最短二进制编码为：0.001

目录

2.0 简介

2.1 信息论基础知识

信息量、熵、信源编码、决策量、编码冗余

2.2 统计编码

2.2.1 香农-范诺编码

2.2.2 霍夫曼编码

2.2.3 算术编码

2.3 行程长度编码

2.4 词典编码

2.4.1 词典编码的思想

2.4.2 LZ77算法

2.4.3 LZSS算法

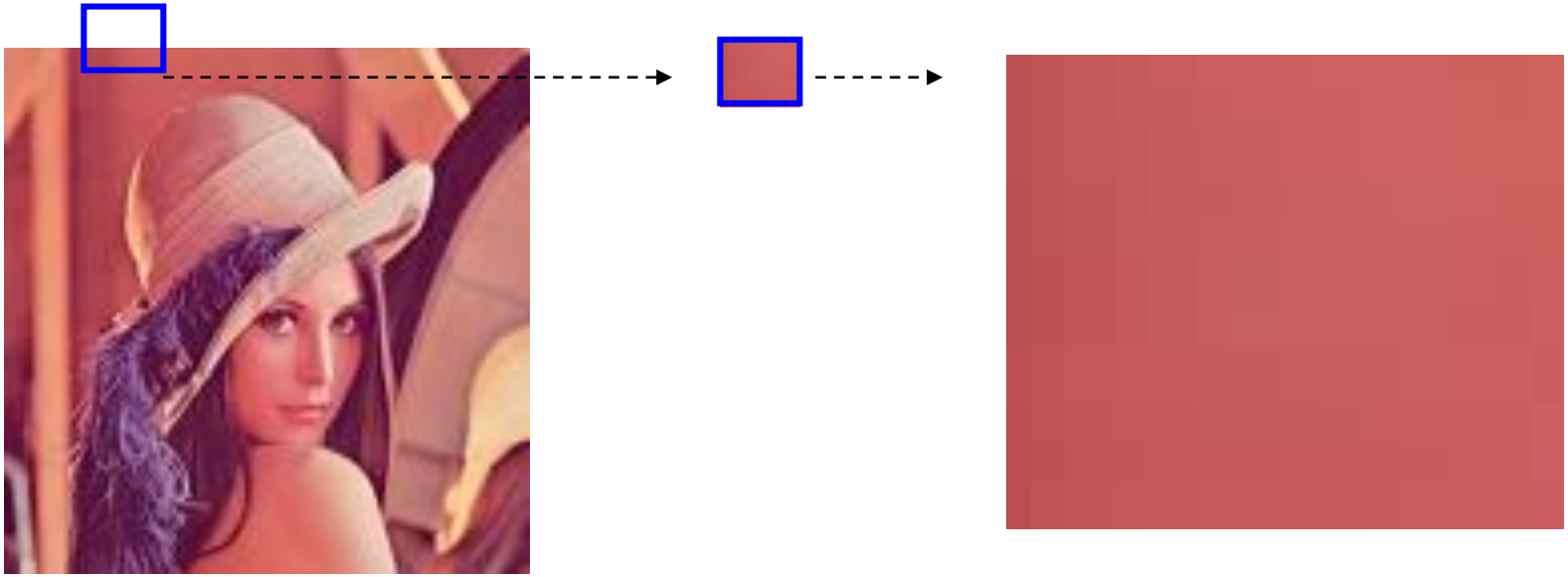
2.4.4 LZ78算法

2.4.5 LZW算法

2.3 行程长度编码(Run Length Encoding, RLE)

RLE编码：

- 利用连续数据单元有相同数值的特点压缩数据



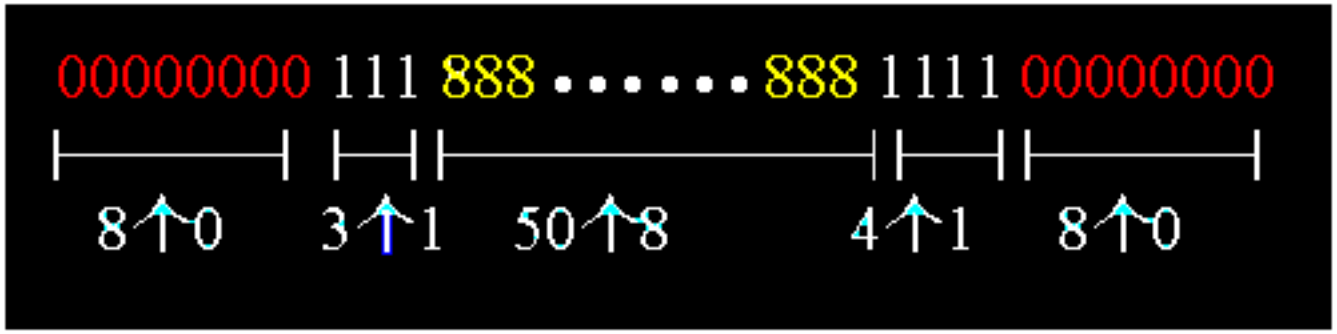
2.3 行程长度编码

- RLE编码：
  - 不需存储每个像素的颜色值，而仅存储一个像素的颜色值以及具有相同颜色的像素数目。

- 行程长度：
  - 具有相同颜色并且是连续的像素数目。

2.3 行程长度编码

- 假定一幅灰度图像，第n行的像素值为：



用RLE编码方法得到的代码为：

8031508418 0。

压缩比 ≈ 7：1

2.3 行程长度编码（特点）

- RLE的压缩比取决于图像本身特点。
  - 如果图像中具有相同颜色的图像块越大，压缩比就越高。反之，压缩比越小。

- 译码时按照与编码时采用的相同规则进行，还原后得到的数据与压缩前的数据完全相同。

2.3 行程长度编码适用范围

- 适用于:
  - 计算机生成的图像、黑白二值图像
- 不适用于:
  - 颜色丰富的自然图像（除非需和其他的压缩编码技术联合应用）、文字

2.3 行程长度编码（适用场景）

- 行程编码多用于黑白二值图像的压缩中。

例如

000000001111111111100000111111

8 12 5 6

被转化为一系列黑串和白串长度的编码：

8C56 (0x)

2.3 行程长度编码

- 对于文本压缩:
  - RLE的效果不好
  - 例如：HoHai->H1o1h1a1i1
  - 除非经常遇到
    - “Ahhhhhhhhhhhhhhhh!”,
    - “Noooooooooo!”.
    - “Gooooooooooooooooooooogle”

目录

2.0 简介

2.1 信息论基础知识

信息量、熵、信源编码、决策量、编码冗余

2.2 统计编码

2.2.1 香农-范诺编码

2.2.2 霍夫曼编码

2.2.3 算术编码

2.3 行程长度编码

2.4 词典编码

2.4.1 词典编码的思想

2.4.2 LZ77算法

2.4.3 LZSS算法

2.4.4 LZ78算法

2.4.5 LZW算法

板凳宽，扁担长。
扁担没有板凳宽，
板凳没有扁担长。

2.4.1 词典编码

- 在很多应用中，信源的输出包含了很多重复的样式（recurring patterns)。

- 对于这种信源，可以用一个列表、或者词典来记录重复出现的样式。

- 词典编码有静态和自适应两种。
  - 大多自适应的方法来根源于Ziv和Lempel在1977年(LZ77)和1978年(LZ78)的两篇文章。

静态词典编码

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

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

$$a\ b\ r\ a\ c\ a\ d\ a\ b\ r\ a$$

2.4.1 词典编码

- **通用编码技术**
  - 不需要知道统计特性

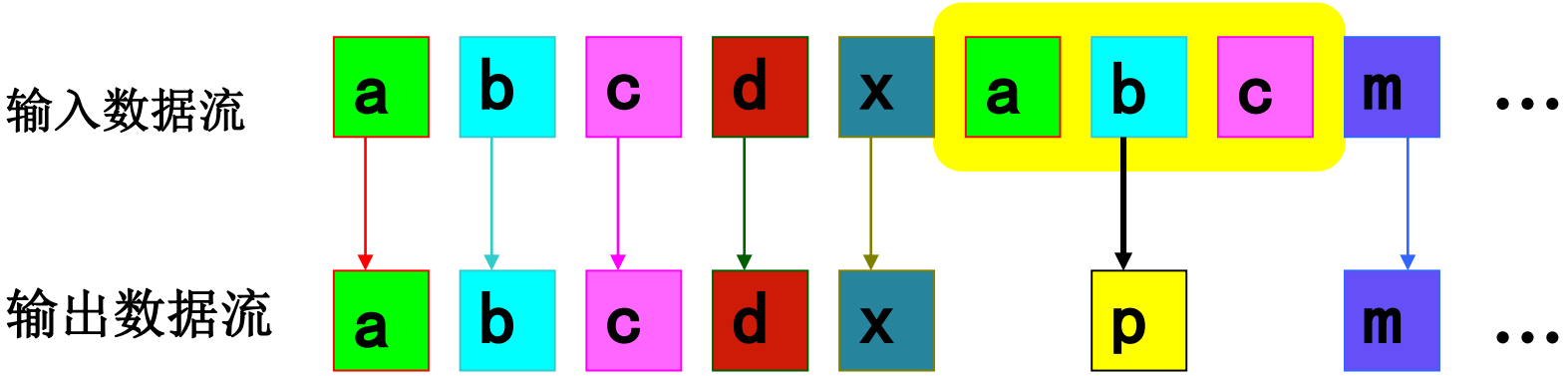
- **利用数据本身包含许多重复字符串**

  例：“吃葡萄不吐葡萄皮,不吃葡萄倒吐葡萄皮”，用简单代号代替这些字符串,实现压缩。

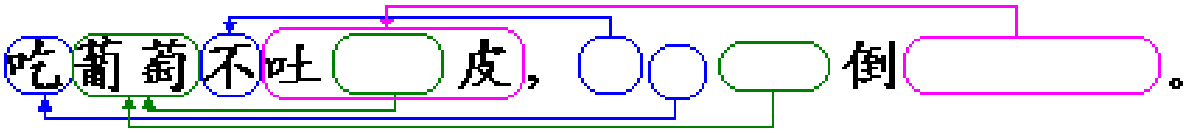
- **实用的词典编码算法的核心**
  - 如何动态地形成词典，以及如何选择输出格式以减小冗余

2.4.1 词典编码(分类)

第一类词典编码思想

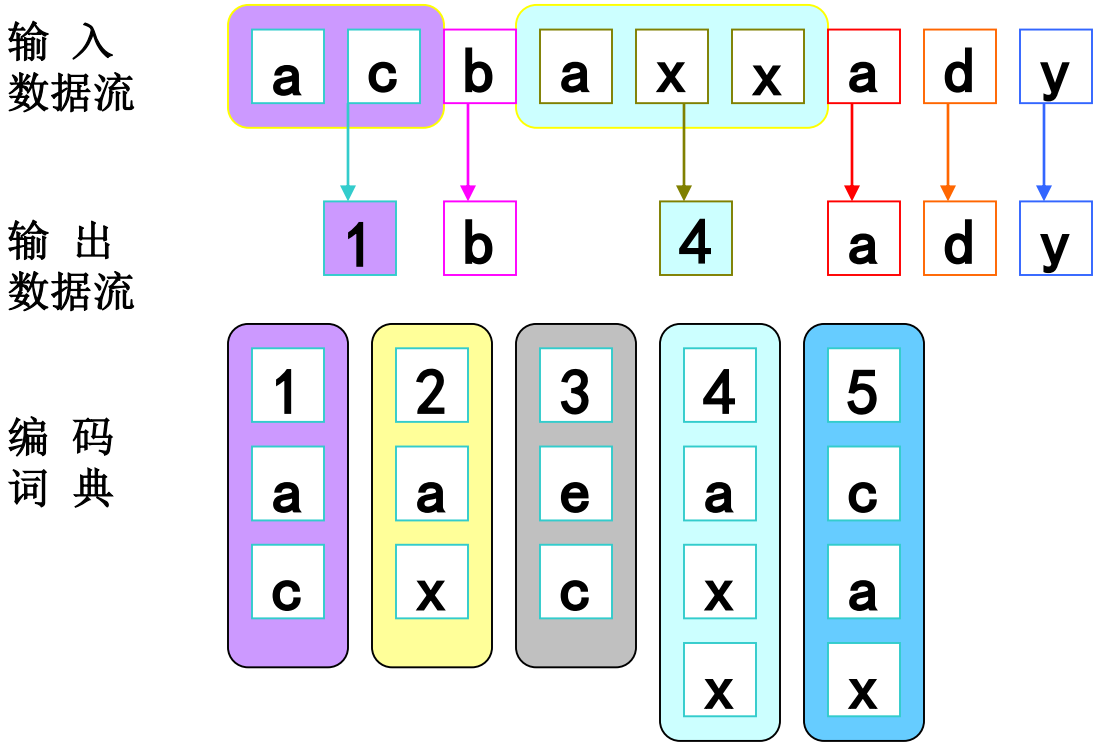


第一类词典法的想法是企图**查找**正在压缩的字符序列是否在**以前输入**的数据中出现过，然后用已经出现过的字符串替代重复的部分，它的输出仅仅是指向早期出现过的字符串的“指针”。



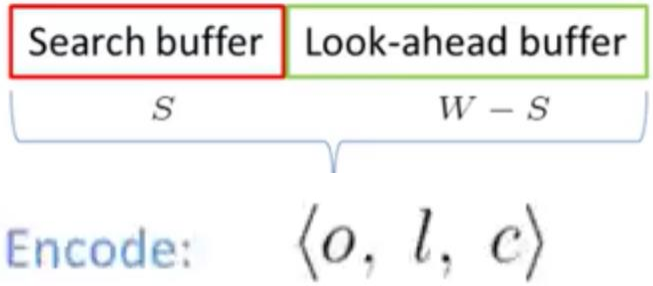
2.4.1 词典编码(分类)

- 第二类词典编码思想



第二类算法的想法是企图从输入的数据中**创建一个“短语词典**(dictionary of the phrases)”，这种短语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。

2.4.2 第一类词典编码-LZ77



- 输出：

<位移，匹配长度，下个字符的码字>

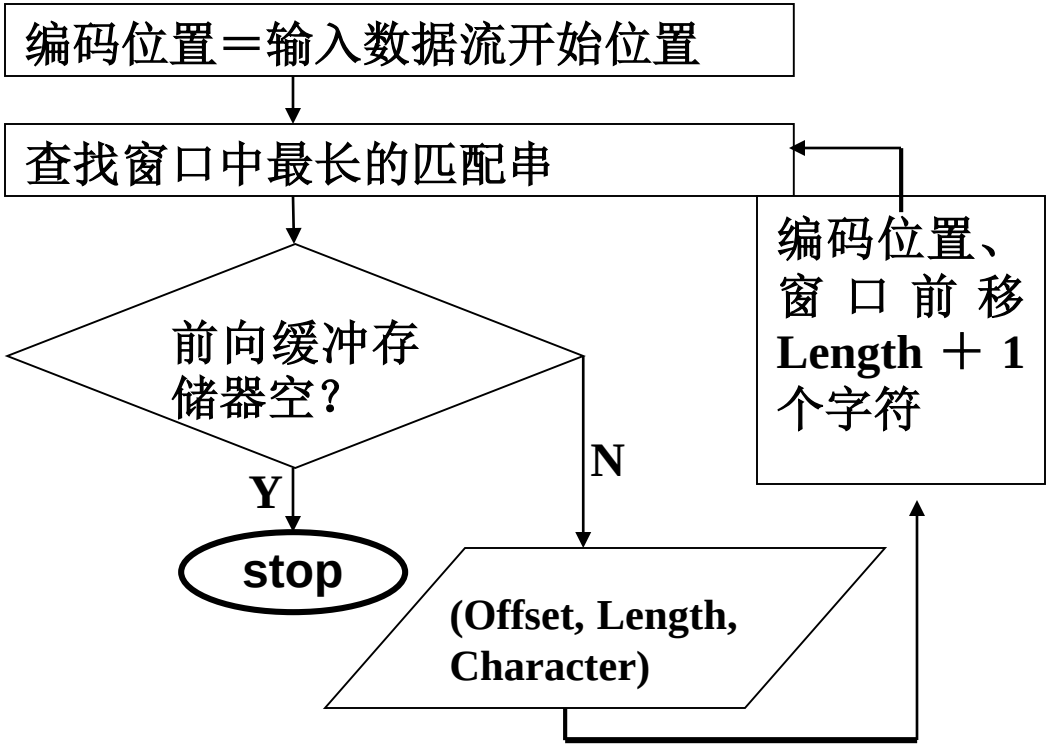
- 窗口前移*l+1*个字符

Bits needed (using fixed length codes)

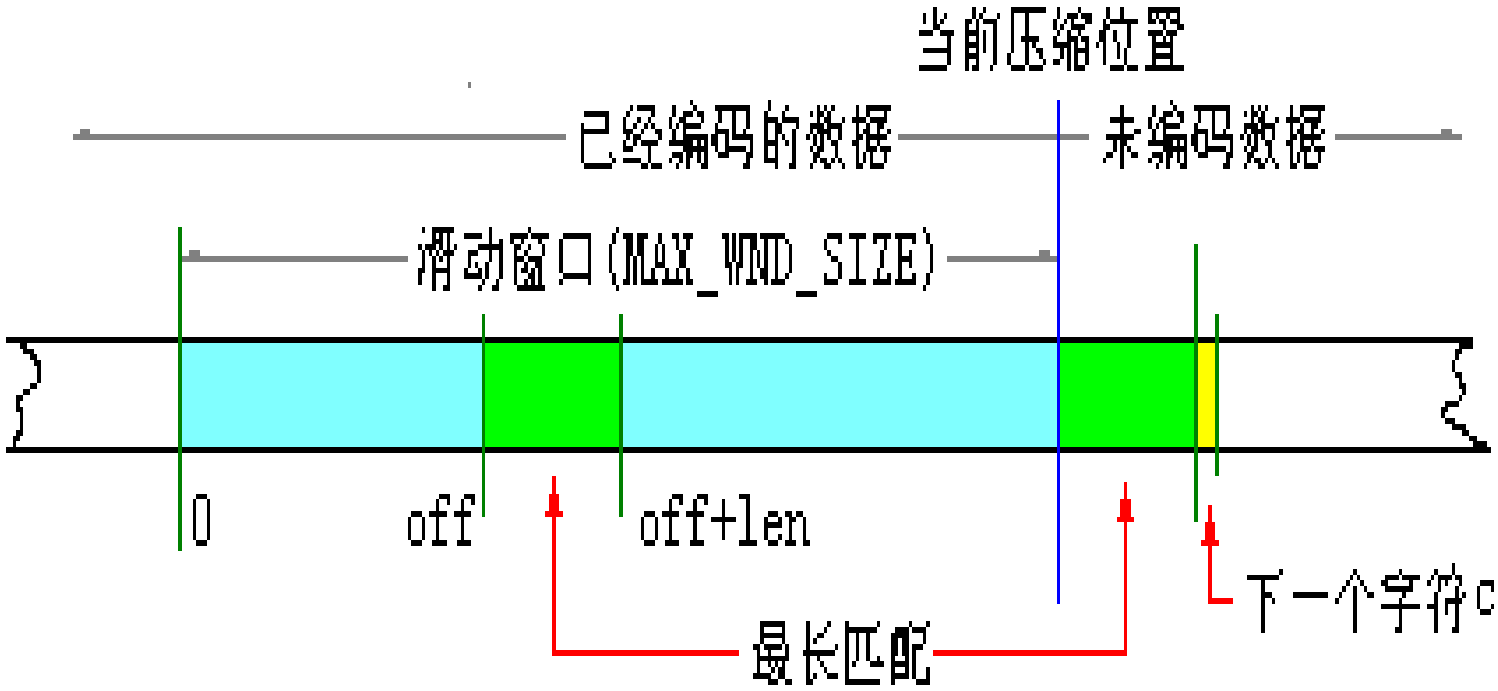
$$\lceil \log_2 S \rceil + \lceil \log_2 W \rceil + \lceil \log_2 |\mathcal{A}| \rceil$$

2.4.2 第一类词典编码-LZ77

- 核心：查找从前向缓冲存储器开始的最长的匹配串



2.4.2 第一类词典编码-LZ77



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

2.4.2 LZ77（例1）

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

Example: Decoding

c a b r a c a

2.4.2 LZ77（例2）

位置	1	2	3	4	5	6	7	8	9	10
字符	A	A	B	C	B	B	A	B	C	A

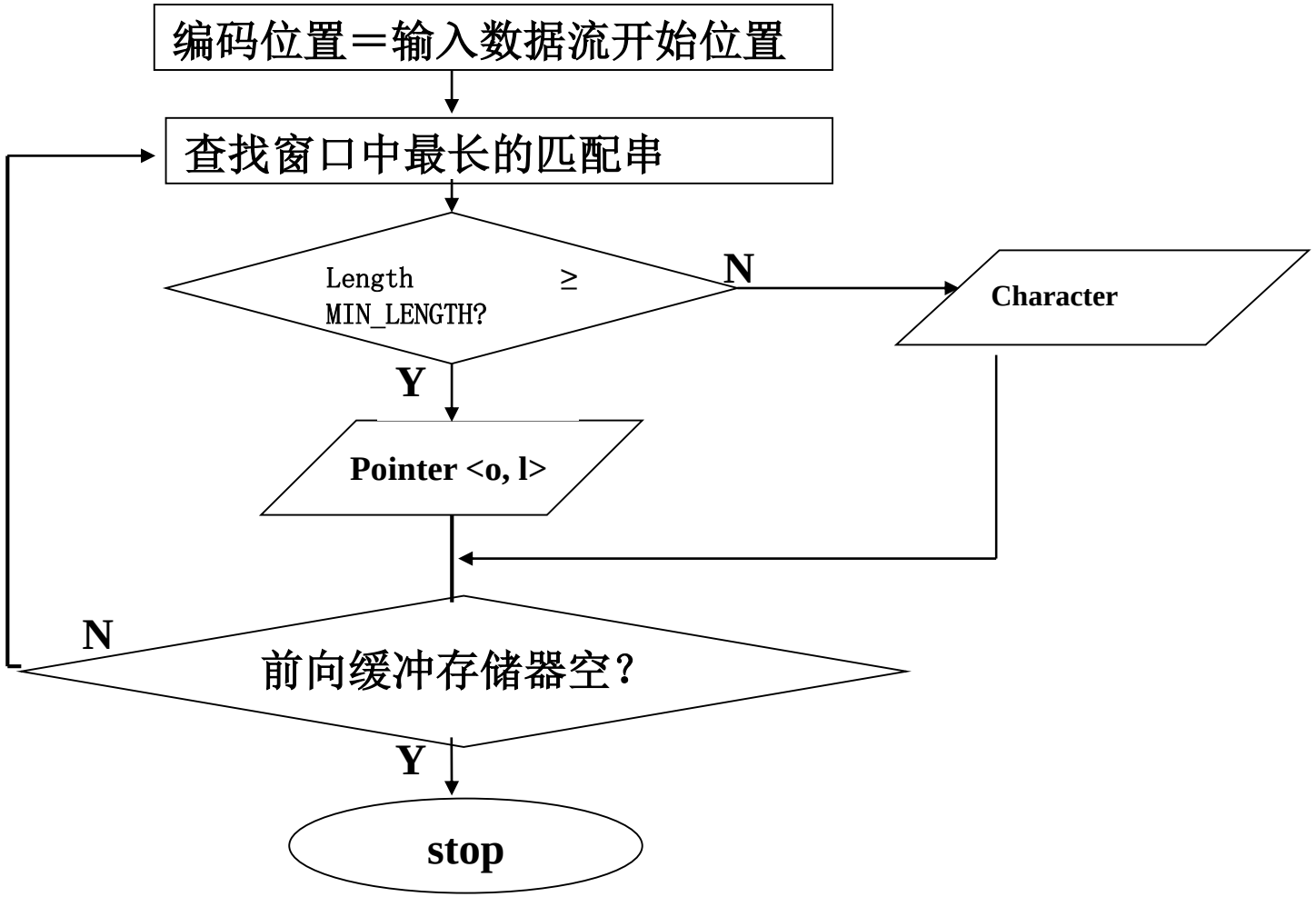
步骤	位置	匹配串	字符	输出
1	1	——	A	(0,0,A)
2	2	A	B	(1,1,B)
3	4	——	C	(0,0,C)
4	5	B	B	(2,1,B)
5	7	ABC	A	(5,3,A)

2.4.3 第一类词典编码-LZSS

LZ77通过输出真实字符解决了在窗口中出现没有匹配串的问题，但这个解决方案包含有冗余信息。冗余信息表现在两个方面，一是空指针，二是编码器可能输出额外的字符，这种字符是指可能包含在下一个匹配串中的字符。

LZSS算法以比较有效的方法解决这个问题，它的思想是如果匹配串的长度比指针本身的长度长就输出指针，否则就输出真实字符。另外要输出额外的标志位区分是指针还是字符。

2.4.3 第一类词典编码-LZSS



2.4.3 LZSS（例1）

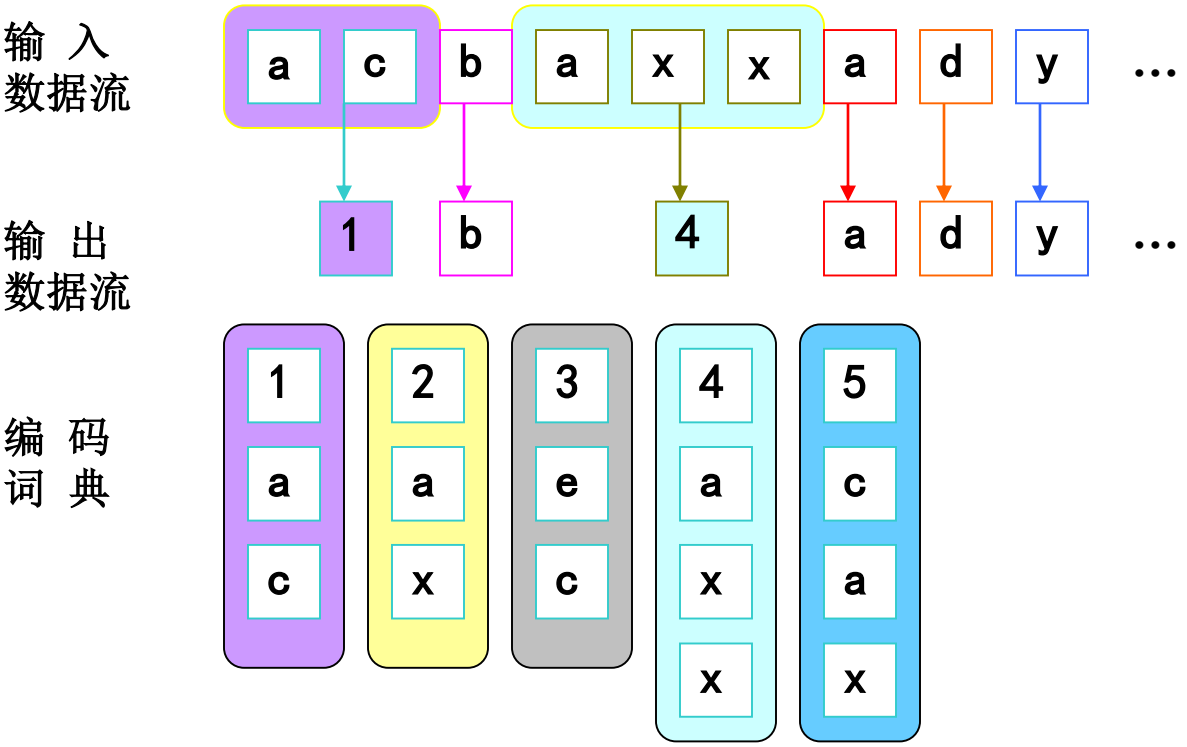
位置	1	2	3	4	5	6	7	8	9	10	11
字符	A	A	B	B	C	B	B	A	A	B	C

Length≥ MIN_LENGTH ?

MIN_LENGTH＝2

步骤	位置	匹配串	输出
1	1	——	A
2	2	A	A
3	3	——	B
4	4	B	B
5	5	——	C
6	6	BB	(3,2)
7	8	AAB	(7,3)
8	11	C	C

词典编码(第二类词典编码思想)



- 从输入的数据中**创建一个“短语词典**（dictionary of the phrases)”。
- 编码过程中遇到已在词典中的“短语”时，输出词典中的短语的“索引号”，而不是短语本身。

词典编码(第二类词典编码历史)

- 第二类词典编码
  - J.Ziv和A.Lempel在1978年首次发表了介绍这种编码方法的文章。(LZ78)
  - 在此基础上，Terry A.Weltch在1984年发表了改进这种编码算法的文章，因此称为LZW(Lempel-Ziv Walch)压缩编码，首先在高速硬盘控制器上应用了这种算法。

2.4.4 第二类词典编码-LZ78

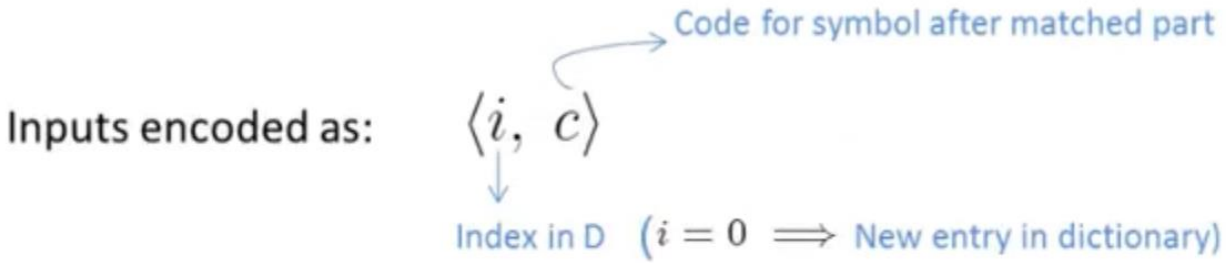
- LZ78的编码思想:
  - 不断地从字符流中提取新的字符串(String)，通俗地理解为新“词条”，然后用“代号”也就是码字(Code word)表示这个“词条”。
  - 对字符流的编码就变成了用码字(Code word)去替换字符流(Char stream)，生成码字流(Code stream)，从而达到压缩数据的目的。

- LZ77利用滑动窗口里的内容作为字典，找最长子串; LZ78动态构建字典

2.4.4 LZ78

- 不依赖于搜索窗口、建立了一个专门的词典

- 编码端和解码端都需要建立这个词典，并且两个词典保持一致。

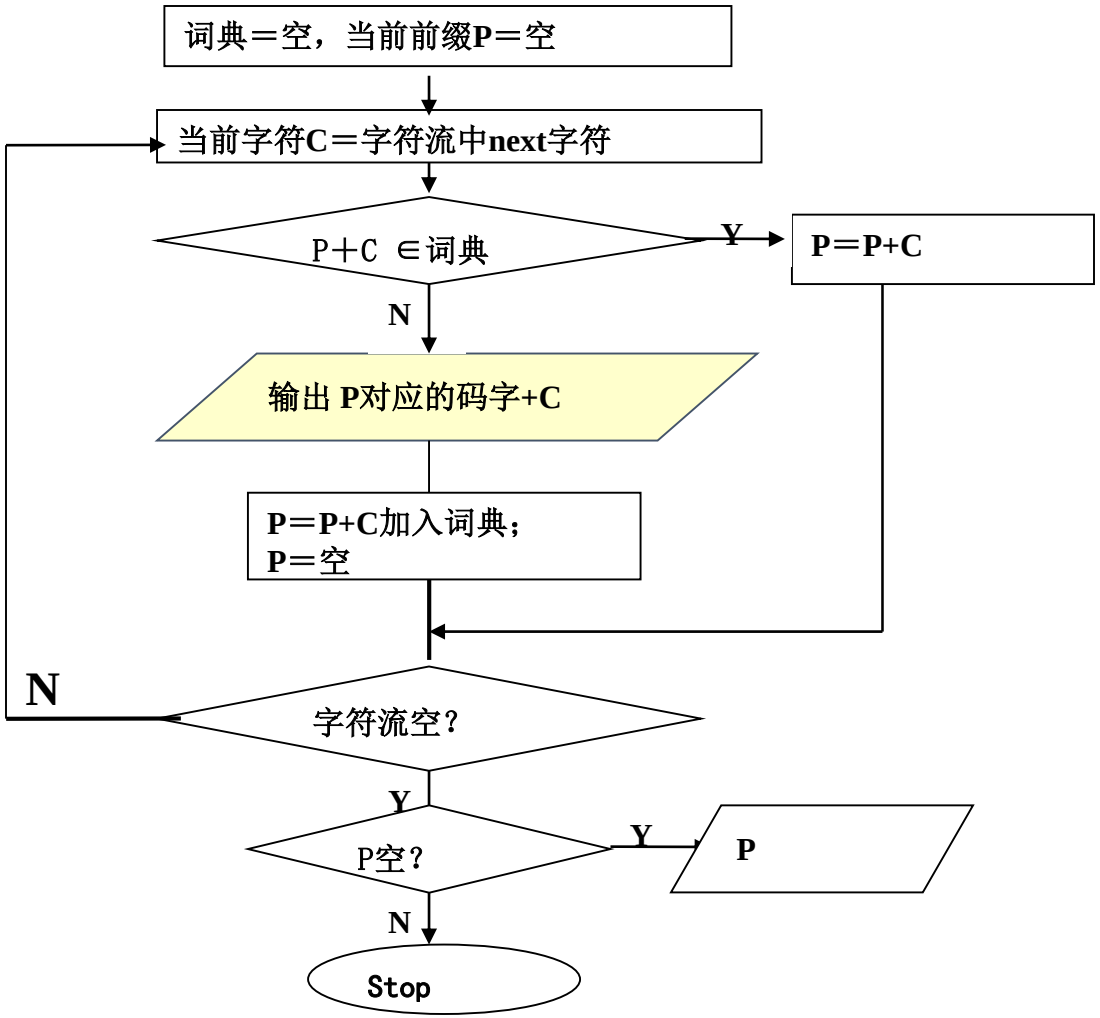


码字-字符(W,C)对，每次输出一对到码字流中，与码字W相对应的缀-符串(String)用字符C进行扩展生成新的缀-符串(String)，然后添加到词典中。

2.4.4 LZ78(例1）

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

步骤	位置	词典	输出
1	1		
2	2		
3	3		
4	5		
5	8		



2.4.4 LZ78(例2）

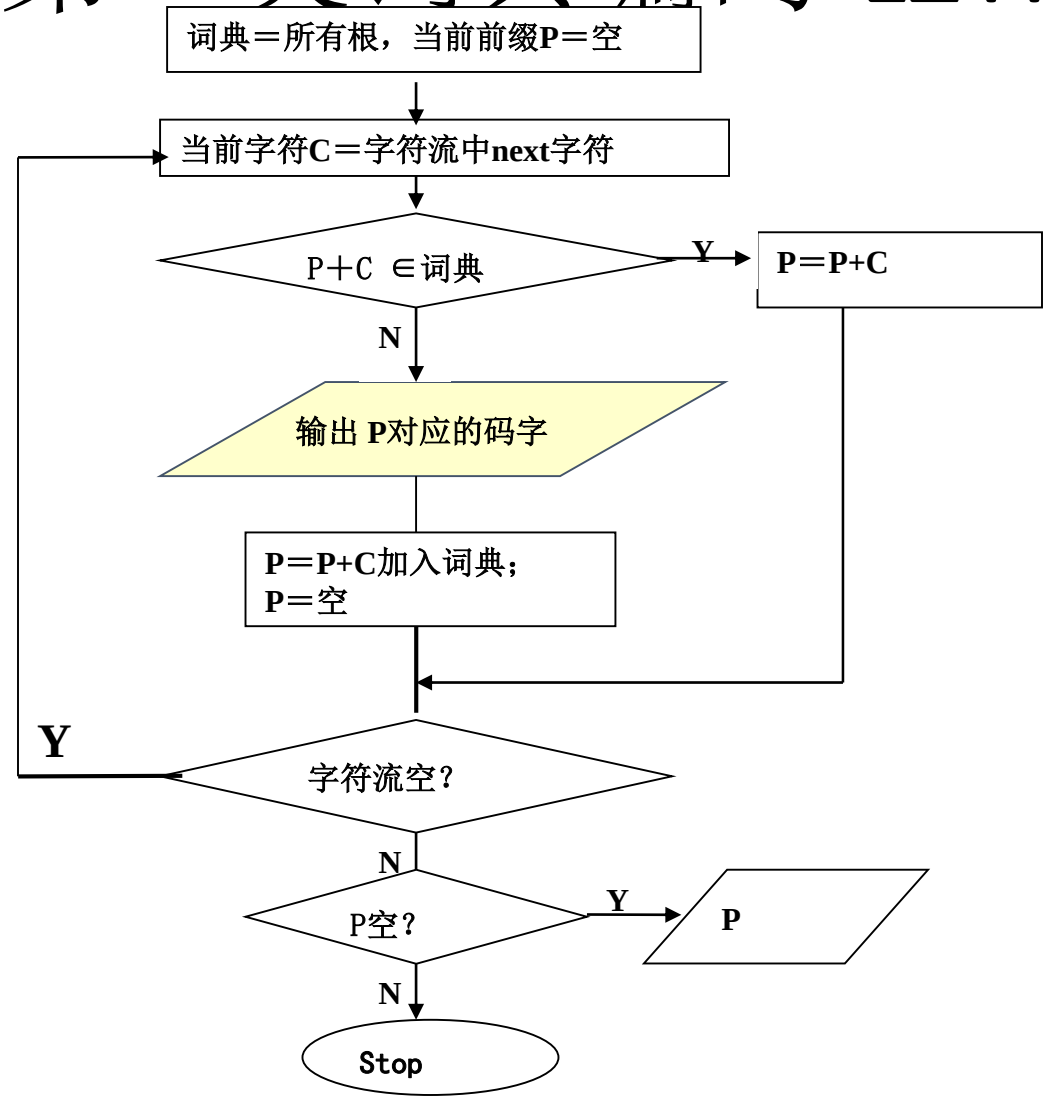
b a w w a / b a w w a /

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/

2.4.5 第二类词典编码-LZW

- LZW算法
  - 只输出代表词典中的字符串(String)的码字(code word)。开始时词典包含可能在字符流出现中的所有单个字符。
  - 每个编码步骤开始时都使用一字符前缀(one-character prefix)，至少可在词典中找到长度为1的匹配串。

2.4.5 第二类词典编码-LZW



2.4.5 LZW编码（例1）

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

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

2.4.5 LZW解码（例1）

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

总结

- 无损压缩
- 熵编码
- 行程长度编码
- 词典编码

作业题：香农-范诺(Shannon- Fano)编码

现有7个待编码的符号，她们的概率如下。计算该符号集的:（1）Shannon-Fano编码；（2）平均码长；（3）“x1 x3 x6 x4”编码结果。

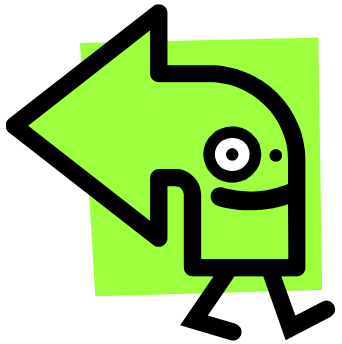
Character	Probability
X1	0.1
X2	0.05
X3	0.2
X4	0.15
X5	0.15
X6	0.25
X7	0.1

作业

- 教科书第43页
  - Shannon-Fano编码，课后题 2.4, 2.5, 2.6
- 提交时间：11月7号（周四）

END

第2章 数据无损压缩



图像无损压缩

- **图像差分编码方法**
  - 给定一个原始图像$I(x, y)$, 可以使用如下操作来获取差分图像 $d(x, y)$

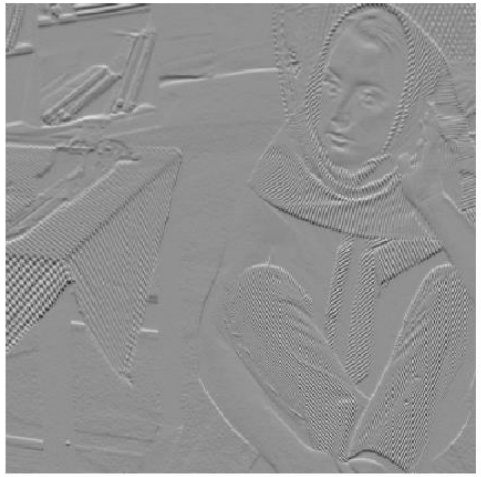
$$d(x, y) = I(x, y) - I(x - 1, y)$$

  - 或使用如下离散2-D 拉普拉斯操作

$$d(x, y) = 4\,I(x, y) - I(x, y - 1) - I(x, y + 1) - I(x+1, y) - I(x - 1, y)$$

“Barb”
原始图像

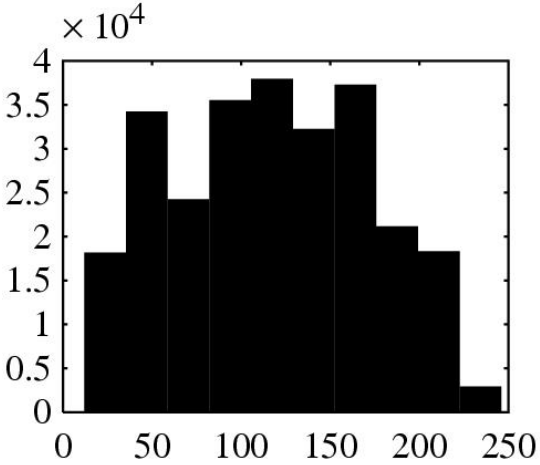
(a)



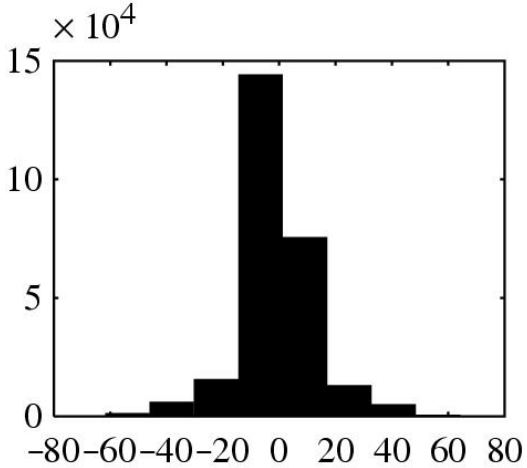
(b)

“Barb”
差分图像

“Barb”
原始图像
直方图



(c)



(d)

“Barb”
差分图像
直方图

- 由于图像中存在的空间冗余，差分图像通常直方图较窄、熵更小

无损JPEG压缩

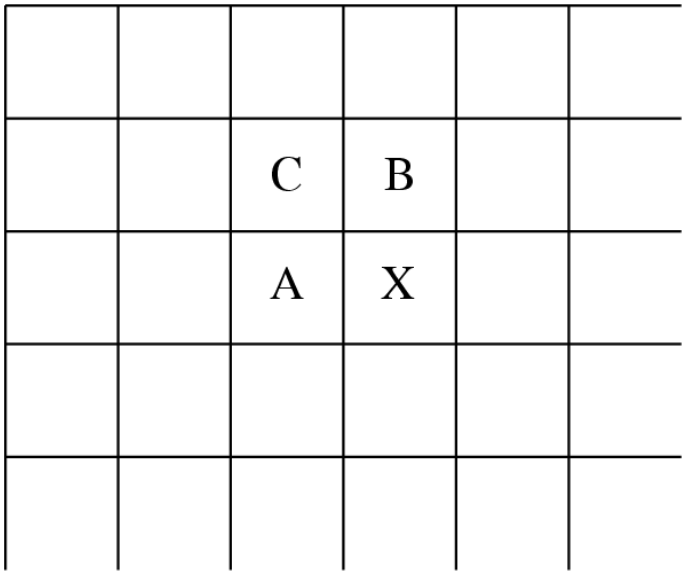
- 无损JPEG压缩: JPEG图像压缩的特殊情况

- 预测方法

1. **差分预测**: 预测器可最多使用3个友邻像素值来预测前像素。预测方法有7种。

2. **编码**: 编码器比较实际像素值与预测值，对预测插值采用例如Huffman编码的无损压缩技术进行编码。

无损JPEG压缩



Predictor	Prediction
P1	A
P2	B
P3	C
P4	A + B - C
P5	A + (B - C) / 2
P6	B + (A - C) / 2
P7	(A + B) / 2

- **Fig. 7.17**: Neighboring Pixels for Predictors in Lossless JPEG.

无损压缩方法对比

Compression Program	Compression Ratio			
	Lena	Football	F-18	Flowers
Lossless JPEG	1.45	1.54	2.29	1.26
Optimal Lossless JPEG	1.49	1.67	2.71	1.33
Compress (LZW)	0.86	1.24	2.21	0.87
Gzip (LZ77)	1.08	1.36	3.10	1.05
Pack(Huffman coding)	1.02	1.12	1.19	1.00