无损压缩算法

讲师：肖俊

软件与技术学院

内容

1、信息论简介与基础

•背景

•数据压缩方案

•信息论基础

1.1 背景

•越来越多的数据以数字形式存在
- •图书馆、博物馆、政府
- •需无损失地存储

•例如 - 对1.2亿个索书号进行编码
- •每个条目需要一个27位的数字， $2^{27} > 120\mathrm{M}$
- •进行压缩以减少所需的位数

•不同的数据出现的频率不同
- •为更频繁出现的数据分配更少的位数
- •VCL - 可变长度编码

•无损编码
- •压缩和解压缩过程均不会导致信息丢失

1.2 数据压缩方案

•定义：压缩比
- •压缩比 $= \mathrm{B0}/\mathrm{B1}$
  - •压缩前的比特数为 B0
  - •压缩后的比特数为 B1

•压缩比必须大于1.0；- 压缩比越高，无损压缩方案越好

通用数据压缩方案

1.3 信息论基础

•信息源的熵
- •字母表为 $S = \{s1, s2, \ldots, sn\}$

$$-\eta = \mathrm{H}(\mathrm{S}) = \sum_{i=1}^{n} p_i \log_2 \frac{1}{p_i} = -\sum_{i=1}^{n} p_i \log_2 p_i$$

- • p_i 是字母表 S 中符号 s_i 出现的概率

• $\log_2 P_i$ 表示字符所包含的信息量（自信息）

1.3 信息论基础

•例如：手稿中 **n** 的出现概率为1/32，因此

- •信息量为5比特
- •字符串nnn编码需要15比特

•什么是熵?
- •系统无序程度的一种度量——熵越大，无序程度越高

1.3 信息论基础

•示例：
- •假设一个系统有4种状态结果，每种结果的概率为 $1/4$：

$$4 \times \frac{1}{4} \times \log_{1/4} 2 = 2 bits$$

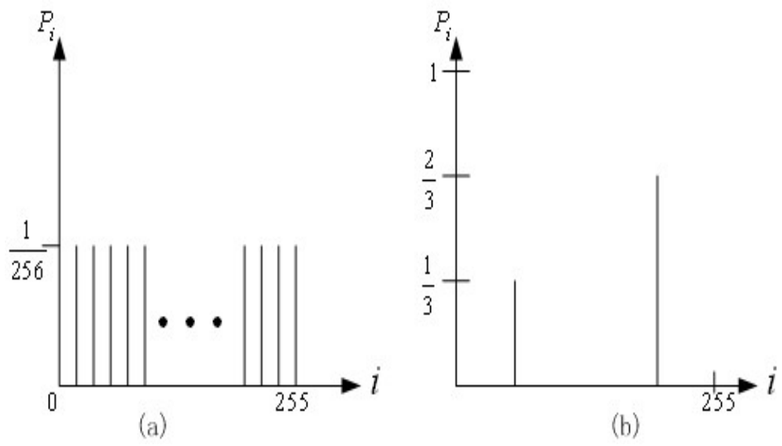
- •如果一个状态的概率为 $1/2$，则其他三个状态的概率为 $1/6$：

$$\frac{1}{2} \times \log_2^2 + 3 \times \frac{1}{6} \times \log_2^6 = 1.795 < 2 bits$$

- •出现次数最多的状态意味着需要发送的比特数更少

•熵的定义——将频繁出现的符号识别为短码字——变长编码

1.3 信息论基础



两幅灰度图像的直方图

1.3 信息论基础

•上述两幅图像的熵

•图像a的熵为：

$$-\eta = \sum_{i=0}^{255} \frac{1}{256} \cdot \log_2^{256} = 8$$

•图像 b 的熵为：

$$\eta = \frac{1}{3} \cdot \log \frac{3}{2} + \frac{2}{3} \cdot \log \frac{\frac{3}{2}}{2}$$

$$= 0.33 \times 1.59 + 0.67 \times 0.59 = 0.92$$

•当概率分布较为平坦时，熵较大；当概率分布更集中时，熵较小。

1.3 信息论基础

•从式(7.3)可以看出：熵 η 是各项 $\log_2 \frac{1}{n}$ 的加权和；因此，它表示信源 S 中每个符号所含的平均信息量。

•熵 η 规定了对 S 中每个符号进行编码所需的平均比特数的下限，即

$$\eta \leq \bar{l} \qquad (7.5)$$

$\bar{l}$ - 编码器生成的码字的平均长度（以比特为单位）。

2、无损编码算法

•游程编码

•变长编码

•基于字典的编码

•算术编码

2.1 游程编码

•游程编码（RUN - LENGTH CODING）

•最简单的数据压缩形式之一

•基本思想：

•如果信息源的符号倾向于形成连续的组，则对这样一个符号和该组的长度进行编码，而不是对每个符号单独编码

•示例：使用游程编码可以有效地对二值图像进行编码

•二维游程编码通常用于对二值图像进行编码。

2.1 游程编码

10	10	10	10	10	10	10	
10	10	10	10	12	12	12	12
10	10	10	10	12	12	12	12
0	0	0	10	10	10	0	0
5	5	5	0	0	0	0	0
5	5	5	10	10	9	9	10
5	5	5	4	4	4	0	0
0	0	0	0	0	0	0	0

8x8灰度图像

第一行 :10,8

第二行 : 10, 5, 12, 3

Third row :10,5,12,3

第四行 : 0, 3, 10, 3, 0, 2

第五行 : 5, 3, 0, 5

第六行 :5,3,10,2,9,2,10,1

第七行 : 5, 3, 4, 3, 0, 2

第八行 :0,8

水平游程长度编码

2.1 游程编码

输入序列：

$0, 0, -3, 5, 1, 0, -2, 0, 0, 0, 0, 2, -4, 3, -2, 0, 0, 0, 1, 0, 0, -2$

游程长度序列：

(2,-3)(0,5)(0,1)(1,-2)(4,2)(0,-4)(0,3)(0,-2)(3,1)(2,-2)

•减少需要编码的样本数量

•实现简单

2.2 可变长度编码

•基本思想

•熵表示信息源中的信息量

•可变长度编码是最著名的熵编码方法之一

•香农 - 法诺算法

– 霍夫曼编码

•自适应霍夫曼编码

2.2 可变长度编码

香农 - 范诺算法

•由贝尔实验室的香农和麻省理工学院的罗伯特·范诺开发

•自顶向下方式

•根据符号出现的频率计数对符号进行排序

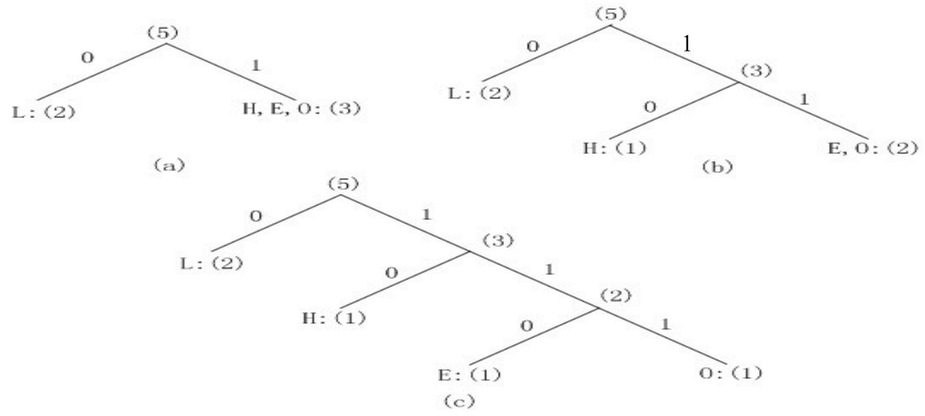
•递归地将符号分成两部分，每部分的计数大致相同，直到所有部分都只包含一个符号

•实现上述过程的一种方法是构建一棵二叉树

2.2 变长编码

示例：Hello

符号	H E L O
计数	1 1 2 1



2.2 可变长度编码

•该示例的熵：

$$0.4 \times 1.32 + 0.2 \times 2.32 + 0.2 \times 2.32 + 0.2 \times 2.32 = 1.92$$

•对.....执行香农 - 费诺算法的一个结果

"Hello"：平均比特数 $10/5 = 2$

符号	计数	$\text{Log}_2\ P_i^{-1}$	代码	使用的比特数
L	2	1.32	0	2
H	1	2.32	10	2
E	1	2.32	110	3
O	1	2.32	111	3
比特总数：10				

2.2 可变长度编码

霍夫曼编码

• 由大卫·A·霍夫曼于1952年首次提出

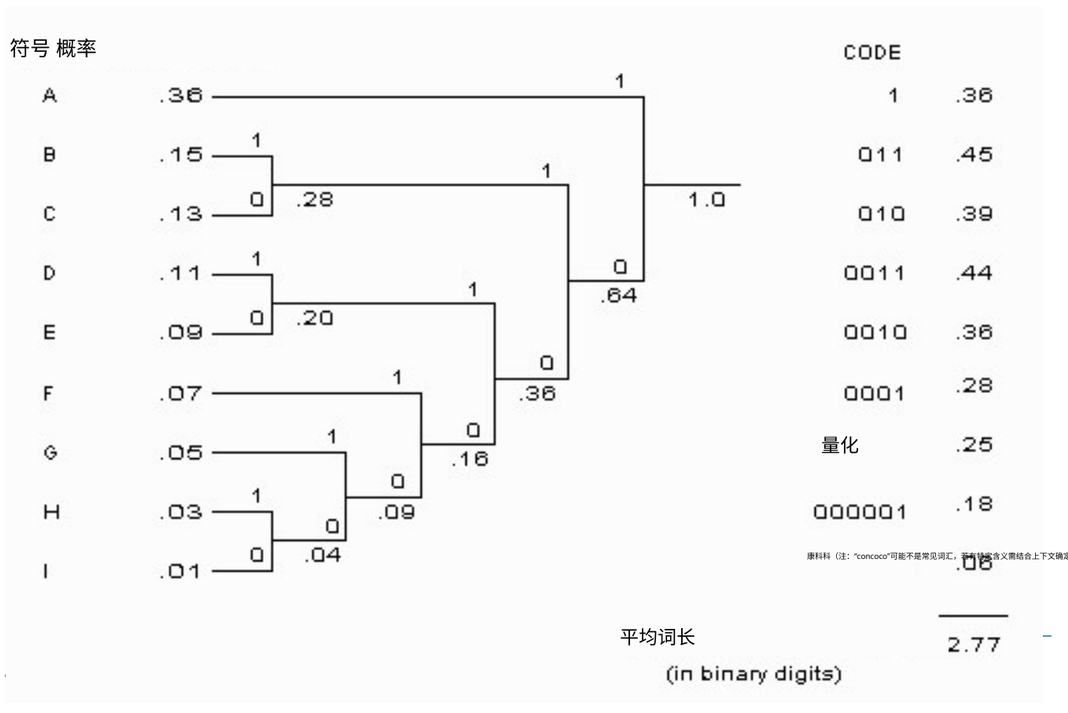
•应用于传真、JPEG、MPEG等应用程序

•自底向上的方式：

•初始化：将所有符号按其频率计数排序后放入列表中 - 重复操作直到列表中只剩一个符号： - 从列表中选取频率计数最低的两个符号，形成一个以这两个符号为子节点的霍夫曼子树，并为它们创建一个父节点

•将子节点的频率计数之和赋给父节点，并将其插入列表以保持顺序。 - 从列表中删除子节点 - 根据从根节点的路径为每个叶子节点分配一个码字

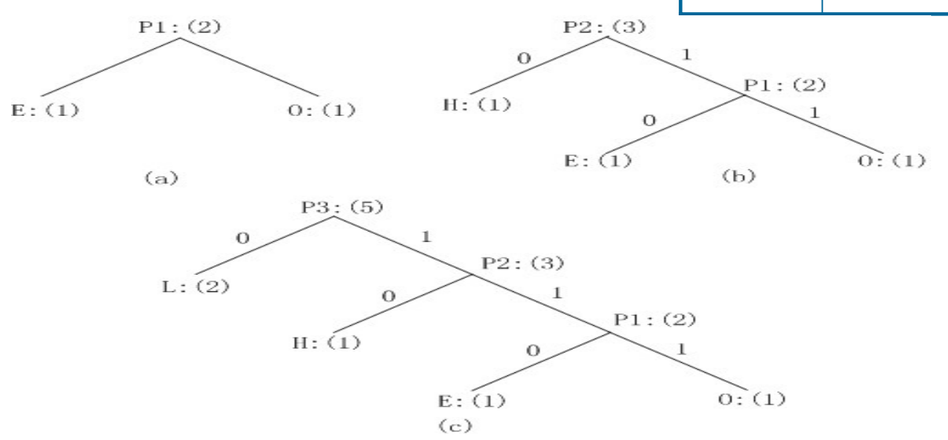
2.2 变长编码



2.2 变长编码

示例：对“Hello”进行霍夫曼编码

符号	H E L O
计数	1 1 2 1



2.2 可变长度编码

•对于上述示例，霍夫曼编码生成的编码结果与香农 - 范诺算法相同

•另一个示例

– A:(15)，B:(7)，C:(6)，D:(6) 和 E:(5)

- 香农 - 范诺编码需要 89 位；- 霍夫曼编码需要 87 位

2.2 变长编码

霍夫曼编码的特性

1. 唯一前缀特性：没有任何霍夫曼编码是其他霍夫曼编码的前缀 - 避免了解码时的任何歧义。

2. 最优性：最小冗余编码 - 已证明对于给定的数据模型（即给定的、准确的概率分布）是最优的：

• 两个出现频率最低的符号的霍夫曼编码长度相同，仅最后一位不同。

•出现频率较高的符号的霍夫曼编码比出现频率较低的符号的霍夫曼编码更短。

信息源 S 的平均码长严格小于 $\eta + 1$。结合式 (7.5)，我们有：

$$\bar{l} < \eta + 1 \qquad (7.6)$$

2.2 可变长度编码

•扩展霍夫曼编码

•动机：霍夫曼编码中的所有码字都具有整数位长。当 p_i 非常大且 $\log_2 \frac{1}{p_i}$ 接近 0 时，这是浪费的。

•为什么不将几个符号组合在一起，并为整个组分配一个单一的码字呢?

•扩展字母表：对于字母表 $S = \{s1, s2, \ldots, sn\}$，如果将 k 个符号组合在一起，那么扩展字母表为：

$$S^{(k)} = \{\overbrace{s_1 s_1 \ldots s_1}^{k\ symbols}, s_1 s_1 \ldots s_2, \ldots, s_1 s_1 \ldots s_n, s_1 s_1 \ldots s_2 s_1, \ldots, s_n s_n \ldots s_n\}.$$

— 新字母表 $S(\mathrm{k})$ 的大小为 n^{k}。

2.2 可变长度编码

•可以证明，每个符号的平均 # 比特数为：

$$\eta \leq \overline{l} < \eta + \frac{1}{k} \tag{7.7}$$

相较于原始的霍夫曼编码有所改进，但改进不大。

•问题：如果 k 相对较大（例如，$k \geq 3$），那么对于大多数实际应用而言，$\mathrm{n} \gg 1, \mathrm{n}^{\mathrm{k}}$ 意味着一个庞大的符号表——不切实际。

2.2 可变长度编码

自适应霍夫曼编码

•霍夫曼编码需要有关信息源的先验统计知识，而这些知识通常是无法获取的

•即使有统计数据，符号表的传输也可能带来巨大的开销

•自适应算法会在数据流到达时动态地收集和更新统计信息

•概率不再基于先验知识，而是基于到目前为止实际接收到的数据

2.2 变长编码

•自适应霍夫曼编码的基本思想

•初始编码：为符号分配一些初始约定的代码；

•更新树：构建自适应霍夫曼树——增加符号的频率计数；

•更新树的结构。

•编码器和解码器必须使用完全相同的初始编码和更新树程序。

2.2 可变长度编码

自适应霍夫曼编码：随着数据流的到来，统计信息会动态收集和更新。

ENCODER
```
--------
初始代码( );

当未到达文件末尾时
  {
        获取(c);
        encode(c);
        更新树(c);
}
```

DECODER
```
--------
初始代码( );

当未到达文件末尾时
{
        解码(c);
        output(c);
        更新树(c);
}
```


2.2 变长编码

自适应霍夫曼树更新笔记

•节点从左到右、从下到上依次编号。括号中的数字表示计数。

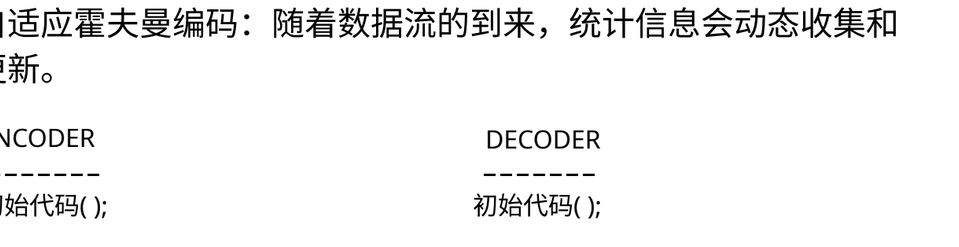
•树必须始终保持其兄弟节点属性，即所有节点（内部节点和叶子节点）按计数递增的顺序排列。

如果即将违反兄弟节点属性，则调用交换过程，通过重新排列节点来更新树。

•当需要进行交换时，计数为 N 的最远节点

与计数刚刚增加到 $N+1$ 的节点进行交换。

2.2 变长编码



(a) 一棵霍夫曼树　(b) 接收到第二个 ' A+ 触发了一次交换

(c - 1) 收到第3个 ' A ' 后需要进行交换　(c - 2) 需要进行另一次交换　(c - 3) 接收到 3rd°A° 后的霍夫曼树

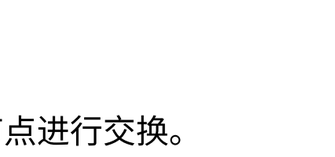
图7.6：自适应霍夫曼树更新的节点交换

2.2 变长编码

另一个示例：自适应霍夫曼编码

•这是为了更清晰地说明更多实现细节。我们确切展示发送了哪些比特，而不是简单说明树是如何更新的。

•附加规则：如果要首次发送任何字符/符号，则必须在其前面加上一个特殊符号NEW。NEW的初始代码为0。NEW的计数始终保持为0（计数从不增加）；因此，在图7.7中它始终表示为NEW:(0)。



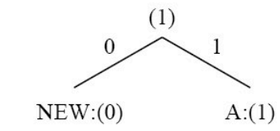
"A"

2.2 变长编码

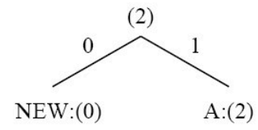
表7.3：使用自适应霍夫曼编码对AADCCDD的初始代码分配。

	初始代码
NEW:	0
A:	00001
B:	00010
C:	00011
D:	00100
	. .
	. .
	. .

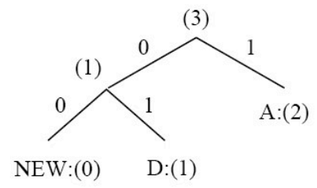
2.2 变长编码



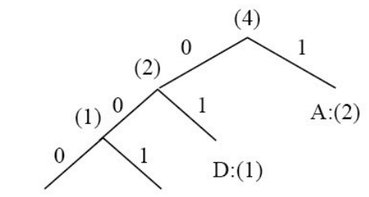
"A"



"AA"

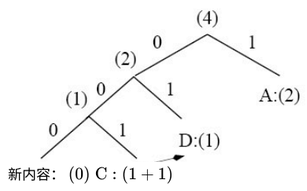


"AAD"

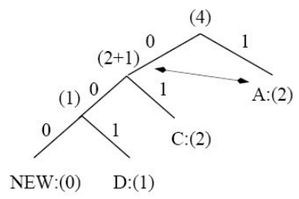


"AADC"

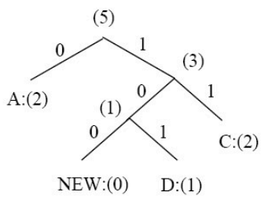
2.2 变长编码



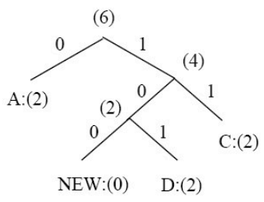
“AADCC”步骤1



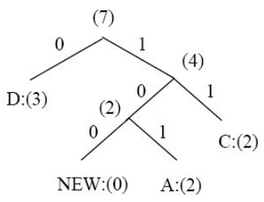
“AADCC”步骤2



“AADCC”步骤3



"AADCCD"



"AADCCDD"

2.2 可变长度编码

符号	NEW	A	A	NEW	D	NEW	C	C	D	D
代码	0	00001	1	0	00100	00	00011	001	101	101

•需要强调的是，在自适应霍夫曼编码过程中，特定符号的代码会发生变化。

例如，在出现AADCCDD之后，当字符D超过A成为出现频率最高的符号时，其代码从101变为0。

2.3 基于字典的编码

•分别由齐夫（Ziv）和伦佩尔（Lempel）于1977年和1978年首次提出

•特里·韦尔奇（Terry Welch）于1984年改进了该技术

•莱姆佩尔 - 齐夫 - 韦尔奇算法（称为LZW压缩）

•它应用于例如UNIX压缩、GIF、调制解调器的V.42 bis协议等

2.3 基于字典的编码

•LZW使用固定长度的码字来表示通常一起出现的可变长度的符号/字符串，例如英语文本中的单词。

•LZW编码器和解码器在接收数据时动态构建相同的字典。

•LZW将越来越长的重复条目放入字典中，如果某个元素已经被放入字典，那么就输出该元素的代码，而不是字符串本身。

2.3 基于字典的编码

算法7.2 - LZW压缩

```
BEGIN
    s = 下一个输入字符;
    当未到达文件末尾时
     {
          c = 下一个输入字符;

          如果 s + c 存在于字典中
          s = s + c;
           else
           {
          输出 s 的编码;
          用新编码将字符串 s + c 添加到字典中;
          s = c;
           }
     }
    输出 s 的编码;
END
```


2.3 基于字典的编码

示例7.2 字符串“ABABBABCABABBA”的LZW压缩

•让我们从一个非常简单的字典（也称为“字符串表”）开始，该字典最初仅包含3个字符，代码如下：

代码	字符串
1	A
2	B
3	C

•现在，如果输入字符串是“ABABBABCABABBA”，LZW压缩算法的工作过程如下：

2.3 基于字典的编码

S	C	输出	代码	字符串
			1	A
			2	B
			3	C
A	B	1	4	AB
B	A	2	5	BA
A	B			
AB	B	4	6	ABB
B	A			
BA	B	5	7	BAB
B	C	2	8	BC
C	A	3	9	CA
A	B			
AB	A	4	10	ABA
A	B			
AB	B			
ABB	A	6	11	ABBA
A	EOF	1		

•输出代码为：124523461。只需发送9个代码，而非14个字符（压缩比 = 14/9 = 1.56）。

2.3 基于字典的编码

算法7.3 LZW解压缩（简单版本）

```
BEGIN
    s = 空;
    当未到文件末尾时
    {
      k = 下一个输入代码;
      条目 = k 对应的字典条目;
      输出条目;
      如果 (s != 空值)
                将字符串 s + 条目[0] 以新代码添加到字典中;
      s = 条目;
    }
END
```

示例 7.3：字符串 "ABABBABCABABBA" 的 LZW 解压缩。解码器的输入代码为 124523461。初始字符串表与编码器使用的相同。

2.3 基于字典的编码

LZW解压缩算法的工作流程如下：

S	K	条目/输出	代码	字符串
			1	A
			2	B
			3	C
NIL	1	A		
A	2	B	4	AB
B	4	AB	5	BA
AB	5	BA	6	ABB
BA	2	B	7	BAB
B	3	C	8	BC
C	4	AB	9	CA
AB	6	ABB	10	ABA
ABB	1	A	11	ABBA
A	EOF			

显然，输出字符串为“ABABBABCABABBA”，这是一个真正无损的结果！

2.3 基于字典的编码

算法7.4 LZW解压缩（修改版）

```
BEGIN
    s = 空;
    当未到文件末尾时
    {
        k = 下一个输入代码;
        条目 = k 对应的字典条目;

        /* 异常处理程序 */
        如果 (entry 为空)
          entry = s + s[0];

        输出 entry;
        如果 (s 不为空)
          将字符串 s＋ entry[0] 以新代码添加到字典中;
        s = 条目;
    }
END
```


2.3 基于字典的编码示例:

输入字符串:"^WED^WEAWEELWEBAWET"。

w	k	输出	鞋垫	斯瓦博尔
NIL	^			
^	W	^	256	^W
W	E	W	257	WE
E	D	E	258	ED
D	^	D	259	D^
^	W			
^W	E	256	260	^WE
E	^	E	261	E^
^	W			
^W	E			
^WE	E	260	262	^WEE
E	^			
E^	W	261	263	E^W
W	E			
WE	B	257	264	WEB
B	^	B	265	B^
^	W			
^W	E			
^WE	T	260	266	^WET
T	EOF	T		

2.3 基于字典的编码

示例（续）：输入字符串:"^WED<256>E<260><261><257>B<260>T".

w	k	输出	索引	符号
	^	^		
^	W	W	256	^W
W	E	E	257	WE
E	D	D	258	ED
D	<256>	^W	259	D^
<256>	E	E	260	^WE
E	<260>	^WE	261	E^
<260>	<261>	E^	262	^WEE
<261>	<257>	WE	263	E^W
<257>	B	B	264	WEB
B	<260>	^WE	265	B^
<260>	T	T	266	^WET

2.4 算术编码

• 算术编码是一种更现代的编码方法，通常优于霍夫曼编码。

• 霍夫曼编码为每个符号分配一个具有整数位长度的码字。算术编码可以将整个消息视为一个单元。

• 消息由半开区间 $[a, b)$ 表示，其中 a 和 b 是 0 到 1 之间的实数。最初，该区间为 $[0, 1)$ 。当消息变长时，区间长度缩短，而表示该区间所需的位数增加。

2.4 算术编码

•基本思想

•算术编码不是将每个字符表示为一个码字，而是用一个包含在 $[0, 1]$ 中的半开区间 $[a, b)$ 来表示整个消息。

•区间 $[a, b)$ 的长度等于该消息的概率。在 $[a, b)$ 中选择一个小数并将其转换为二进制形式作为编码输出。

•每个字符都会缩短区间，因此字符越多，区间就越短。

•随着区间变短，需要更多的比特位来表示该区间

2.4 算术编码

示例：算术编码中的编码

符号	概率	范围
A	0.2	[0, 0.2)
B	0.1	[0.2, 0.3)
C	0.2	[0.3, 0.5)
D	0.05	[0.5, 0.55)
E	0.3	[0.55, 0.85)
F	0.05	[0.85, 0.9)
$	0.1	[0.9, 1.0)

(a) 符号的概率分布。

图7.8：算术编码：对符号“CAEE$”进行编码

2.4 算术编码

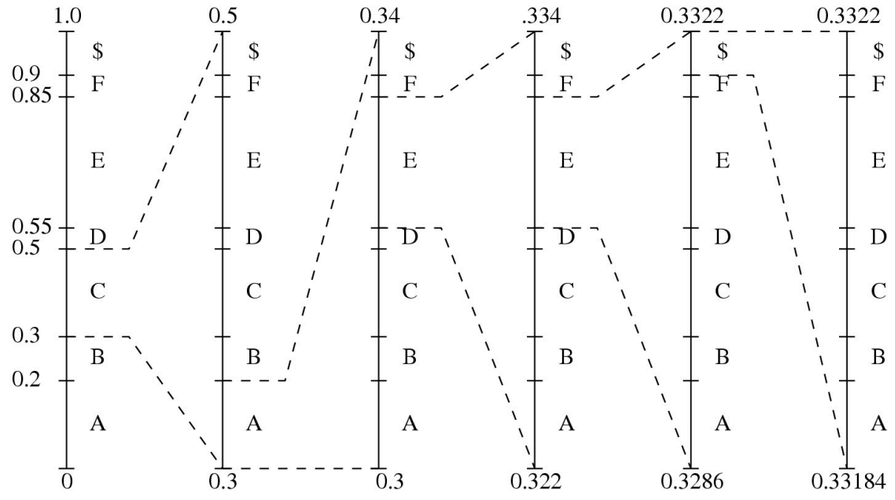


图7.8(b) 范围缩小的图形展示。

2.4 算术编码

示例：算术编码中的编码过程

符号	概率	范围
A	0.2	[0, 0.2)
B	0.1	[0.2, 0.3)
C	0.2	[0.3, 0.5)
D	0.05	[0.5, 0.55)
E	0.3	[0.55, 0.85)
F	0.05	[0.85, 0.9)
$	0.1	[0.9, 1.0)

符号	低	高	范围
	0	1.0	1.0
C	0.3	0.5	0.2
A	0.30	0.34	0.04
E	0.322	0.334	0.012
E	0.3286	0.3322	0.0036
$	0.33184	0.33220	0.00036

(c) 生成新的最小值、最大值和范围。

图 7.8（续）：算术编码：编码符号“CAEES”

2.4 算术编码 过程 7.2 为编码器生成码字

```
BEGIN
    代码 = 0;
    k = 1;
    当 (代码的值) < 下限 时
    {
        将第 k 个二进制小数位赋值为 1
        如果 (代码的值) > 上限
        将第 k 位替换为 0
        k = k + 1;
    }
END
```

对于上述示例，输出为：0.01010101

2.4 算术编码

算法7.6 算术编码解码器

```
BEGIN
    获取二进制代码并转换为
    十进制值 = value(代码);
    Do
    {
        找到一个符号 s 使得
          Range_low(s) <= 值 < Range_high(s);
        输出 s;
        low = Rang_low(s);
        上限 = Range_high(s);
        范围 = 上限 - 下限;
        值 = [值 - 下限] / 范围;
    }
    直到符号 s 为终止符
END
```


2.4 算术编码

表7.5 算术编码：解码符号“CAEES”

值	输出符号	下限	上限	范围
0.33203125	C	0.3	0.5	0.2
0.16015625	A	0.0	0.2	0.2
0.80078125	E	0.55	0.85	0.3
0.8359375	E	0.55	0.85	0.3
0.953125	$	0.9	1.0	0.1

3、无损图像压缩

- 图像差分编码
- 无损JPEG

3.1 无损图像压缩

- **差分编码**
  - 多媒体数据压缩中最常用的压缩技术之一
- 差分编码中数据缩减的基础
  - 数据流中连续符号存在的冗余

3.2 图像的差分编码

给定原始图像 $I(x,y)$，定义差分图像 $d(x,y)$

•使用简单的差分算子

·$\mathrm{d}(\mathrm{x},\mathrm{y}) = \mathrm{I}(\mathrm{x},\mathrm{y}) - \mathrm{I}(\mathrm{x}-1,\mathrm{y})$

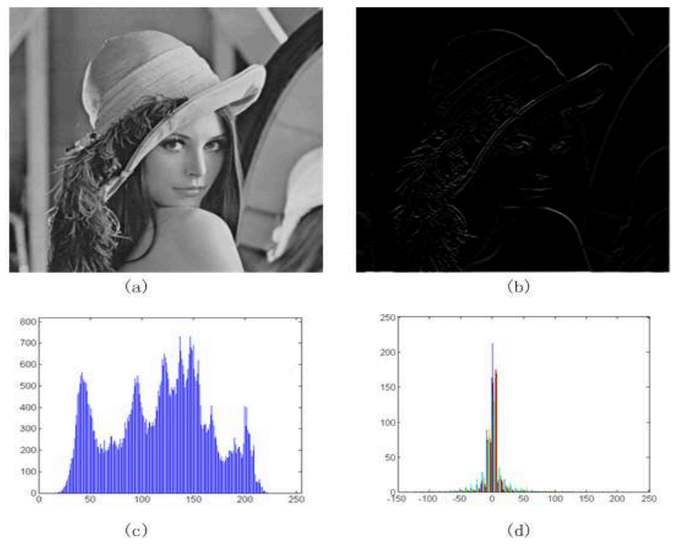
•离散二维拉普拉斯算子

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

•由于普通图像 I 中存在空间冗余，差分图像 d 的直方图会更窄，因此熵更小

•可变长度编码（VLC）——差分图像的比特长度更短——对差分图像进行压缩效果更好

3.2 图像的差分编码



3.3 无损JPEG

•无损JPEG：JPEG图像压缩的一种特殊情况。

•预测方法

1. 形成差分预测：预测器将最多三个相邻像素的值组合起来，作为当前像素的预测值，如图7.10中“X”所示。预测器可以使用表7.6中列出的七种方案中的任意一种。

2. 编码：编码器将预测值与位置‘ X ’处的实际像素值进行比较，并使用我们讨论过的无损压缩技术之一（例如霍夫曼编码方案）对差值进行编码。

3.3 无损JPEG

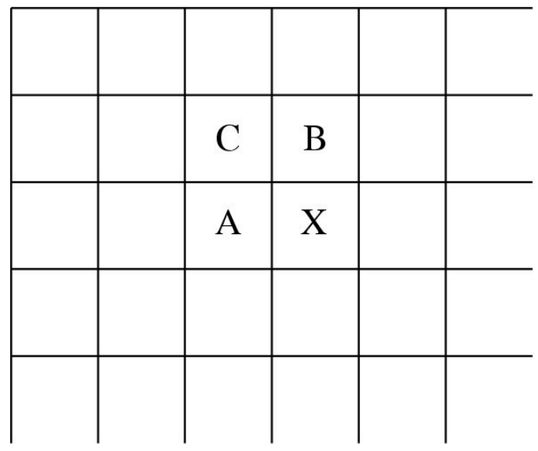


图7.10：无损JPEG中预测器的相邻像素。

•注意：在编码 - 解码循环的解码器端，A、B或C中的任何一个在用于预测器之前都已被解码。

3.3 无损JPEG

表7.6：无损JPEG的预测器

预测器	预测
P1	A
P2	B
P3	C
P4	A + B – C
P5	A + (B – C) / 2
P6	B + (A – C) / 2
P7	(A + B) / 2

3.3 无损JPEG

表7.7：与其他无损压缩程序的比较

压缩程序	压缩比			
	莉娜	足球	F-18	花朵s
无损JPEG	1.45	1.54	2.29	1.26
最优无损JPEG	1.49	1.67	2.71	1.33
压缩（LZW）	0.86	1.24	2.21	0.87
gzip压缩（LZ77）	1.08	1.36	3.10	1.05
gzip -9（最优LZ77）	1.08	1.36	3.13	1.05
打包（霍夫曼编码）	1.02	1.12	1.19	1.00

结束

谢谢！

邮箱：junx@cs.zju.edu.cn