無失真壓縮演算法

PART I

教材出處

- 多媒體概論

  原著: Li Drew, 徐永煜 等譯 高立

  第七章 無失真壓縮演算法
- 資料壓縮

  戴顯權 著，紳藍。
- Introduction to Data Compression,

  Khalid Sayood, Morgan Kaufmann Publishers, Inc.,

目錄

PART I

- 緒論
- 消息理論基礎知識
- 變動碼長編碼
- 霍夫曼編碼
- 霍夫曼編碼的應用

PART II

- 以字典為本的編碼法
- 算術編碼
- 無失真JPEG
- 延伸閱讀

緒論

- **壓縮**：可以有效地降低描述某種資訊所需要的總位元數的處理過程

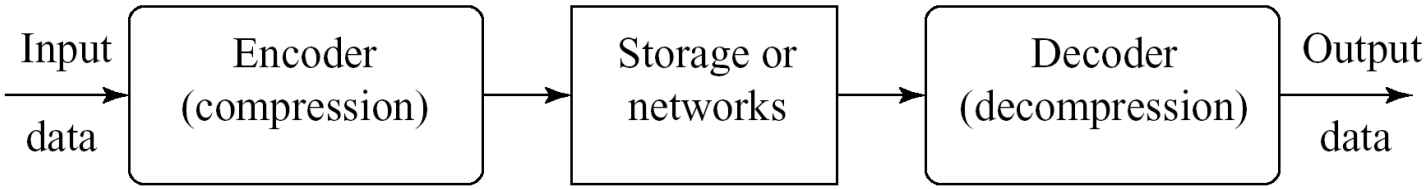


圖7.1 一般的資料壓縮方法

緒論

- 假如壓縮和解壓縮的過程之間沒有任何的資訊損失，這類的壓縮方式就是**無失真**，反之，就是**失真**壓縮
- **壓縮率**：

$$compression\ ratio = \frac{B_0}{B_1} \qquad (7.1)$$

B_0 - 壓縮前的位元個數
B_1 - 壓縮後的位元個數

消息理論基礎知識

- 對於一個以字集(alphabet) $S = \{s_1, s_2, \ldots, s_n\}$ 的資訊來源的熵 (entropy) η 會被定義為

$$\eta = H(S) = \sum_{i=1}^{n} p_i \log_2 \frac{1}{p_i} \quad (7.2)$$

$$= -\sum_{i=1}^{n} p_i \log_2 p_i \quad (7.3)$$

P_i —— s_i在S中發生的機率

$\log_2 \frac{1}{p_i}$ —— 用來表示s_i所包含的訊息量(即薛能所謂的自情報量(self-information))，而這通常也反映著對s_i進行編碼所需要的位元數

灰階強度分布

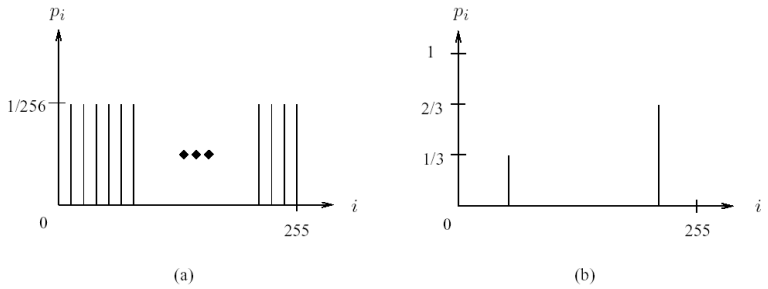


圖7.2 兩張灰階影像的色階分布圖

- 在圖7.2(a)中的影像色階分布圖(histogram) 顯示其像素的灰階值是均勻分布(uniform distribution)，也就是說， $\forall i, p_i = 1/256$ 。
- 因此，這張影像的熵值就是 $\log_2 256 = 8$

Information

- Suppose that we have the source alphabet of q symbols $s_1, s_2, .., s_q$
- For example:
  - 所有的英文句子

    S={a,b,c,…,z,A,B,…Z}
  - 所有的二進位資料

    S={0,1}
  - 一張以n bits 為灰階值的影像

    $S=\{0,1,2,\ldots,2^n-1\}$

Information

DMS (discrete memoryless sources)

- 由它他所產生的連續符號在統計上是獨立事件
- $S=\{s_1, s_2,.., s_q\}$ each with its probability $p(s_i)=p_i$.

$$\sum_{i}^{q} p(s_i) = 1$$

How much information do we get when we receive one of these symbols ?

Information

- A function which measures the amount of information (surprise, uncertainty) of occurrence of receiving the symbol s_i is

$$I(s_i) = \log_2 \frac{1}{p(s_i)} = -\log_2 p(s_i)$$

Entropy

- On the average, over the whole alphabet of symbols s_i, we will get

$$\sum_{i=1}^{q} p_i \log_r \left(\frac{1}{p_i}\right)$$

- Conventionally, we write

$$H_r(S) = \sum_{i=1}^{q} p_i \log_r \left(\frac{1}{p_i}\right) = H_r(P)$$

as the entropy of the signaling system S having symbols s_i and probability p_i.

Entropy

- Example 1 :

$S=\{s_1, s_2, s_3, s_4\}$, $p_1=1/2$, $p_2=1/4$, $p_3= p_4=1/8$.

$$H(S)=\frac{1}{2}\log_2 2+\frac{1}{4}\log_2 4+\frac{1}{8}\log_2 8+\frac{1}{8}\log_2 8$$

$$=\frac{1}{2}+2\cdot\frac{1}{2}+3\cdot\frac{1}{8}+3\cdot\frac{1}{8}=\frac{11}{8}=1.75 bits$$

- Example 2:

$S=\{s_1, s_2, s_3, s_4\}$, $p_1= p_2= p_3= p_4=1/4$.

$$H(S)=\frac{1}{4}\log_2 4+\frac{1}{4}\log_2 4+\frac{1}{4}\log_2 4+\frac{1}{4}\log_2 4=2 bits$$

Entropy

- Example 3:

  Consider the distribution consisting of just two events. (例：擲銅板） Let p be the probability of the first symbol (event). Then, the entropy function is

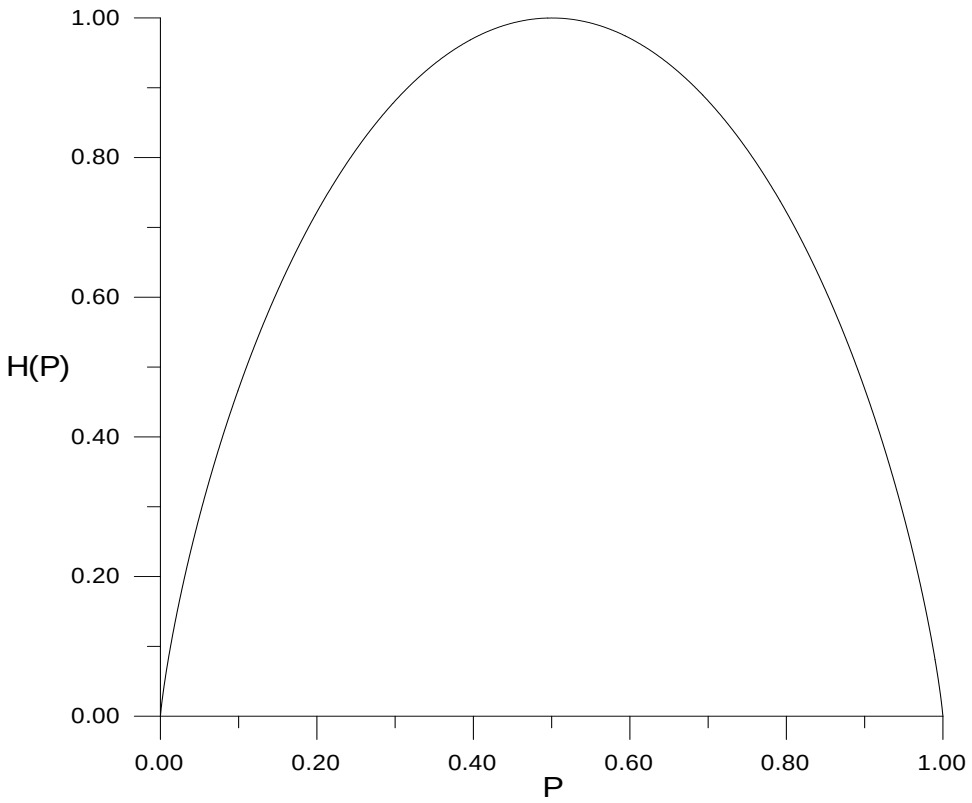
  $H_2(S) = p \log_2(1/p) + (1-p)\log_2[1/(1-p)] = H_2(P)$

- The maximum of $H_2(P)$ occurs when $p = 1/2$.

$$I(s_1) = I(s_2) = -\log_2\left(\tfrac{1}{2}\right)^{-1} = \log_2 2 = 1$$

$$H_2(S) = \tfrac{1}{2} I(s_1) + \tfrac{1}{2} I(s_2) = 1$$

Entropy



- 當所有的符號出現的機率都相等時，$H_2(S) = I(S)$, $H(S)$ 最大。

Entropy and Coding

- **The entropy supplies a lower bound on the average code length L_{avg} for any instantaneous decodable system.**

$$H_2(S) \leq L_{avg}$$

Hint:

$$H_2(S) = \sum_{i=1}^{q} p_i \log_r \left(\frac{1}{p_i}\right)$$

$$L_{avg} = \sum_{i}^{q} l_i \cdot p_i$$

l_i is the code length of s_i.

Entropy and Coding

- Example:

符號	頻率	編碼 1	編碼 2	編碼 3	編碼 4
a_1	1/2	0	0	0	0
a_2	1/4	0	1	10	01
a_3	1/8	1	00	110	011
a_4	1/8	10	11	111	0111
平均碼長 L_{avg}		1.125	1.25	1.75	1.875

$$H_2(S) = \frac{1}{2}\log_2 2 + \frac{1}{4}\log_2 4 + \frac{1}{8}\log_2 8 + \frac{1}{8}\log_2 8 = 1.75 bits$$

Entropy and Coding

- *Unique decodable codes* 唯一解碼

Any given sequence of codewords can be decoded in one, and only one, way.

a_1=0

a_2=1

a_3=00

a_4=11

Does 100 represent

$\{a_2, a_3\}$ or $\{a_2\, a_1 a_1\}$?

Entropy and Coding

符號	編碼 1	編碼 2	編碼 3	編碼 4
a_1	0	0	0	0
a_2	0	1	10	01
a_3	1	00	110	011
a_4	10	11	111	0111
Unique decoded	x	x	√	√
Instantaneous code	-	-	√	x

變動碼長編碼

薛能-法諾演算法——由上而下的策略

1. 根據符元的出現頻率先進行排序
2. 將這些符元分成兩部分，且讓這兩部分各自包含的符元其出現的總次數盡可能相等，然後以遞迴的方式繼續分割每一部份，直到每一部份都僅包含一個符元為止

一個例子：編碼"HELLO"

Symbol	H	E	L	O
Count	1	1	2	1

"HELLO"中各符元出現次數列表

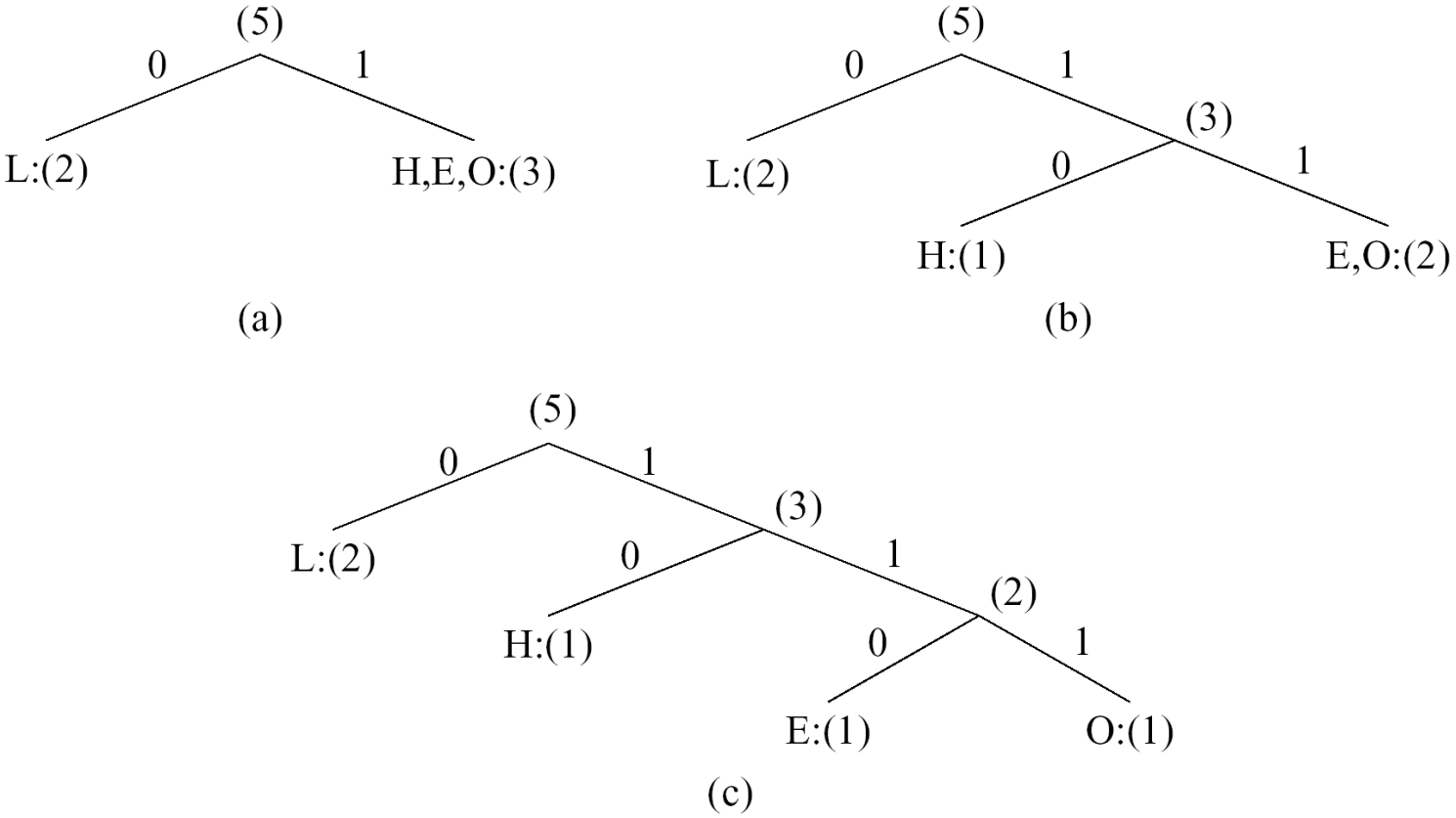


圖7.3 以薛能-法諾演算法對 HELLO 編碼所產生的編碼樹

表7.1 用薛能—法諾演算法對HELLO編碼的結果

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

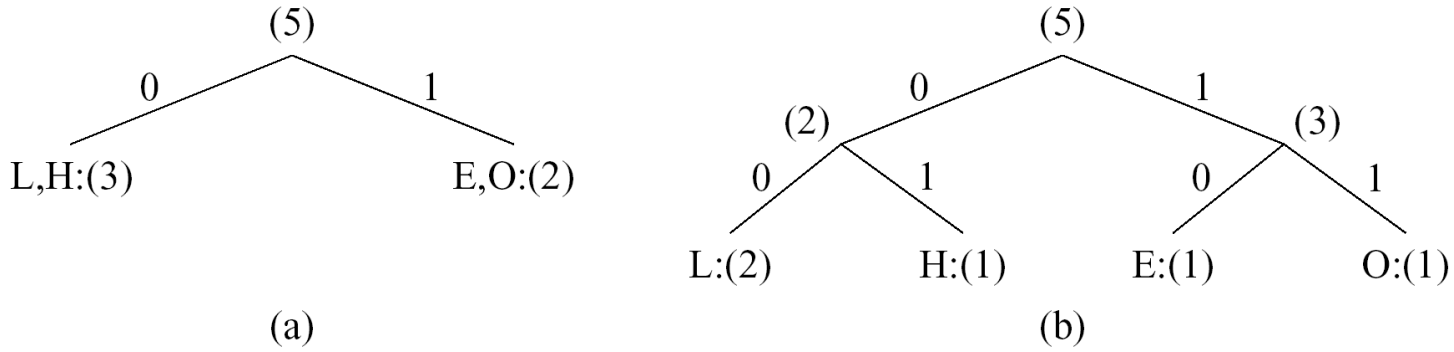


圖7.4 以薛能-法諾演算法對HELLO編碼所產生的另外一種編碼樹

表7.2 薛能—法諾演算法對HELLO編碼所產生的另外一組結果

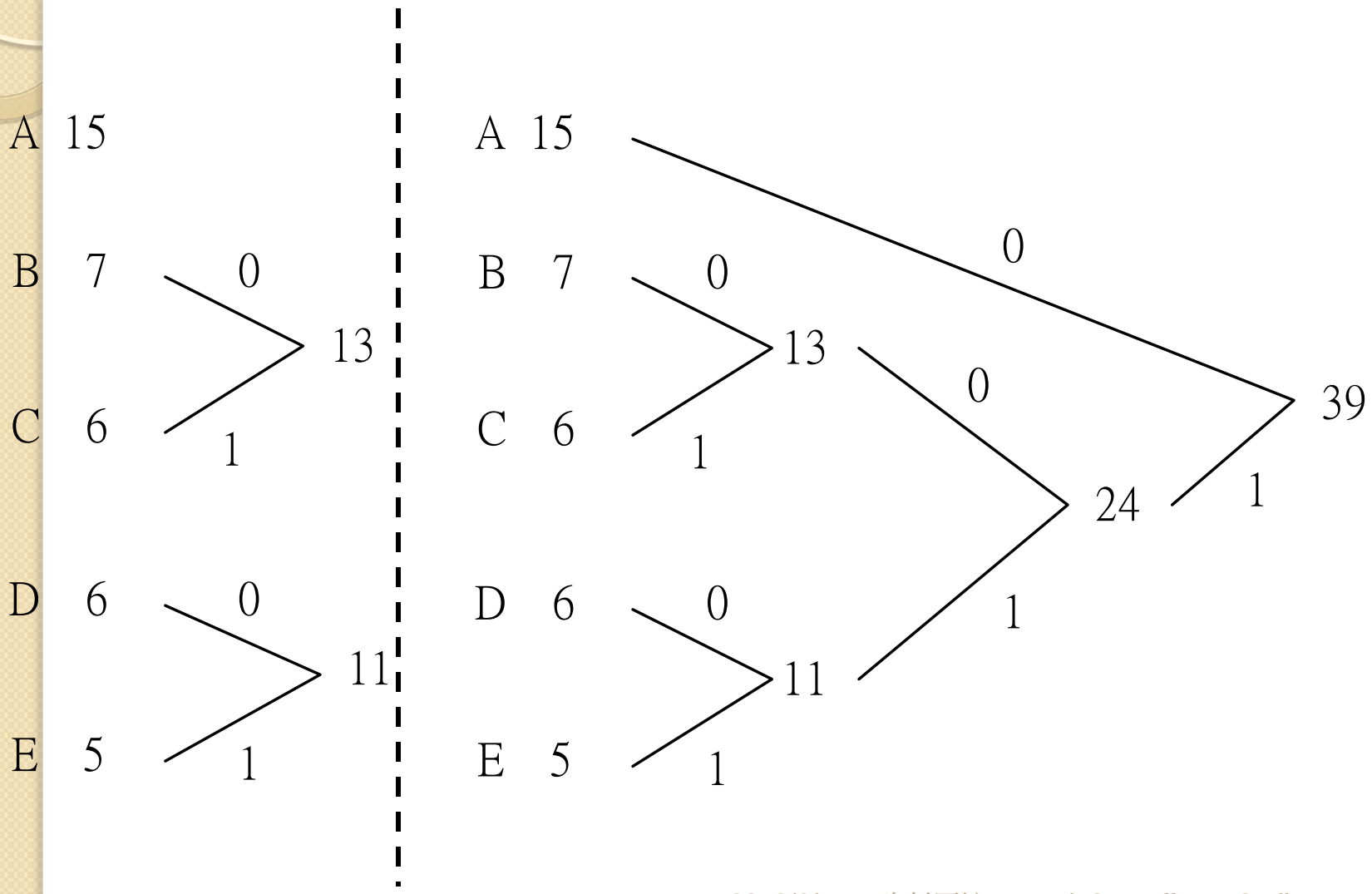
Symbol	Count	$\log_2 \frac{1}{p_i}$	Code	# of bits used
L	2	1.32	00	4
H	1	2.32	01	2
E	1	2.32	10	2
O	1	2.32	11	2
TOTAL number of bits:				10

霍夫曼編碼

演算法7.1 霍夫曼編碼 —— 由下而上的策略

1. 初始化：把所有的符元根據他們的出現頻率排序於串列上
2. 重複下述的過程一直到串列上只剩下一個字符
   1) 在這個串列上取出出現頻率最低的兩個字符，然後把這兩個字符當作是子節點，並在其上建立一個節點當作其父節點
   2) 把子節點的出現頻率加總起來當作是其父節點的出現頻率值，並把此父節點加入到串列中
   3) 把子節點從串列中刪除
3. 根據從根(root)到每一個葉端(leaf)的路徑指定該葉端的字碼

Huffman Coding Example (1)



Huffman Coding

符號	頻率	Huffman code	碼長	總位元數
A	15	0	1	15
B	7	100	3	21
C	6	101	3	18
D	6	110	3	18
E	5	111	3	15

合計 87 bits

Huffman Coding

Huffman Tree Building Algorithm

```
1  procedure huffman (var l : ListPointer ; n : integer);
2  {l is a list of n single node binary trees as described above.}
3  var t : TreePointer ;
4      i : integer;
5  begin
6    for i := 1 to n − 1 do                    {loop n−1 times}
7    begin
8      new(t);                                 {Create a new binary tree}
9      t↑.LeftChild := least (l);              {by combining the trees}
10     t↑.RightChild := least (l);             {with the two smallest weights}
11     t↑.weight := r↑.LeftChild↑.weight + t↑.RightChild↑.weights ;
12     insert (l,t);
13   end;
14 end; {of huffman}
```

least( ) determines a tree in *l* with minimum weight and removes it from *l*.

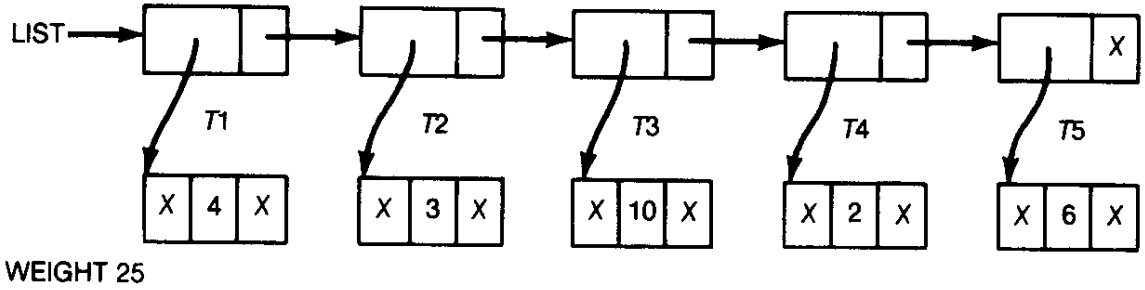
Huffman Coding Example (2)

符號	頻率	Huffman code	碼長	總位元數
A	10			
B	6			
C	4			
D	3			
E	2			

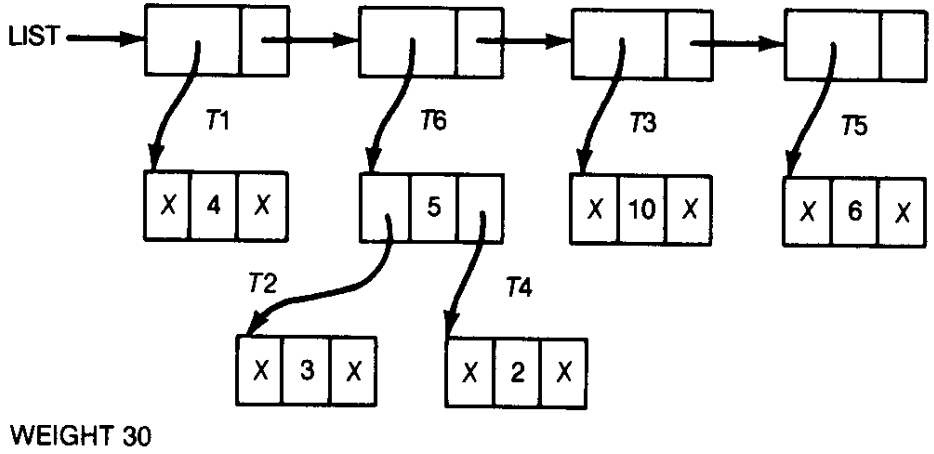
合計　? bits

Huffman Coding Example (2)

- Implementation

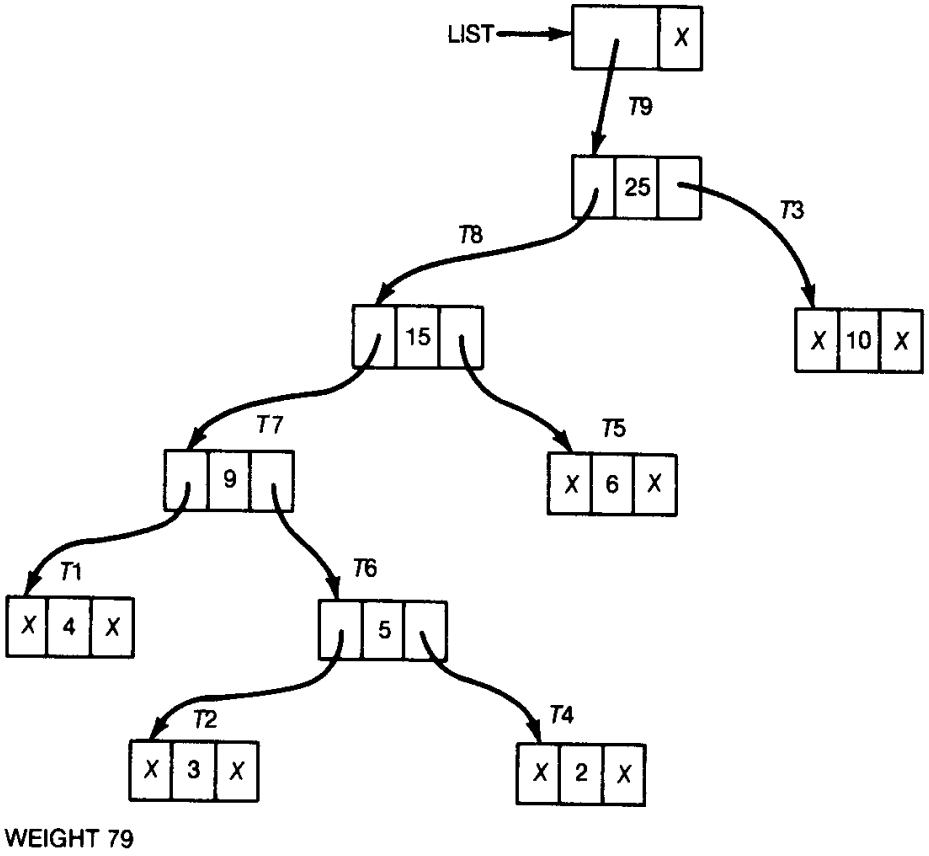


(a) List of Five Records Pointing to Five Initial Trees of the Huffman Algorithm



(b) List after Combining Two Trees with Smallest Values

Huffman Coding Example (2)



(c) Final List and Huffman Tree after Repeated Combinings of Values

Huffman Coding Example (2)

符號	頻率	Huffman code	碼長	總位元數
A	10	1	1	10
B	6	01	2	12
C	4	000	3	12
D	3	0010	4	12
E	2	0011	4	8

合計　54 bits
平均碼長 2.16bits

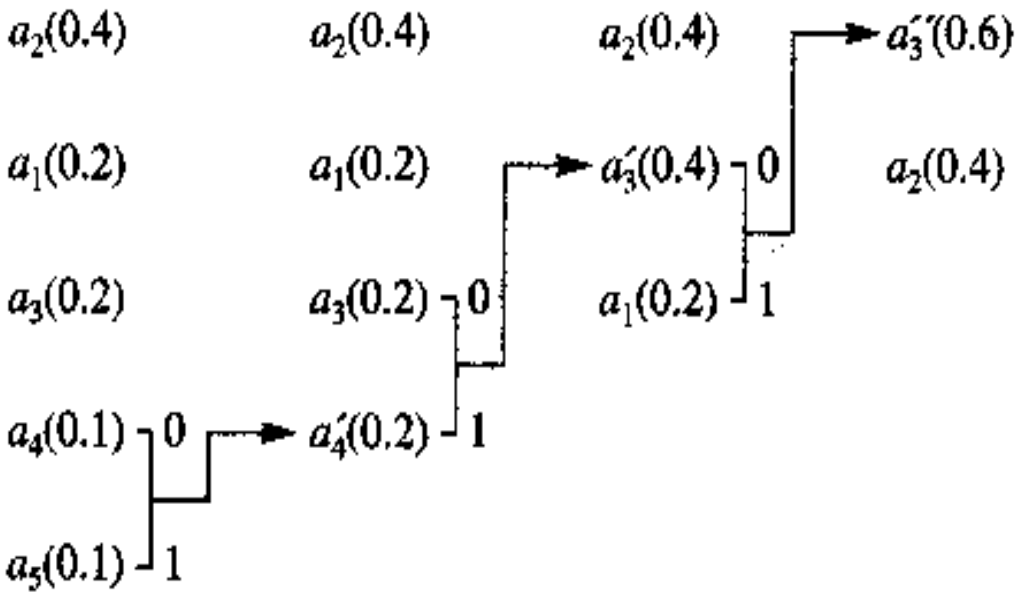
Huffman Coding Example (3)

$$A = \{a_1, a_2, a_3, a_4, a_5\}$$

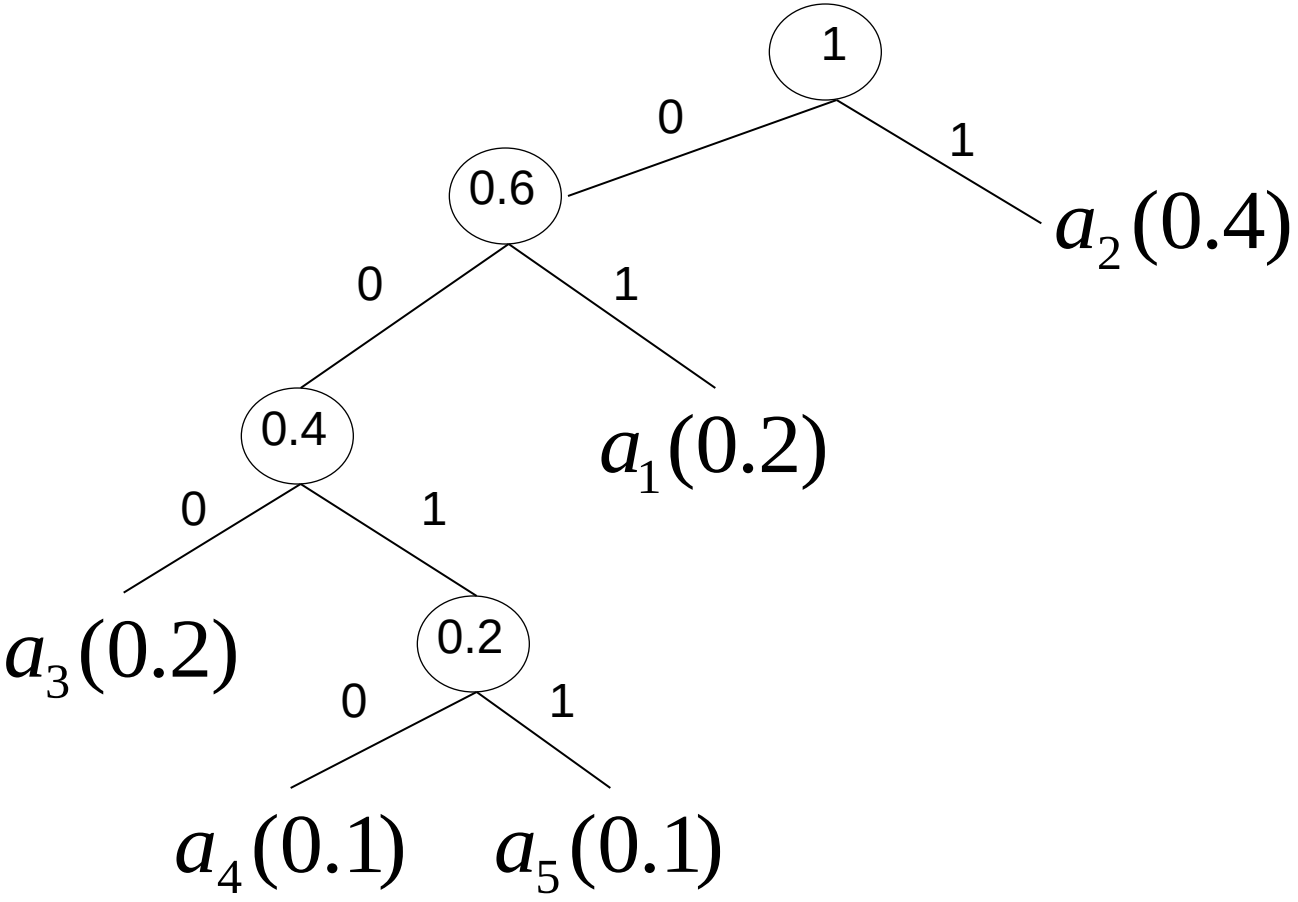
$$P(a_1) = P(a_3) = 0.2$$

$$P(a_2) = 0.4$$

$$P(a_4) = P(a_5) = 0.1$$



Huffman Coding Example (3)



Huffman Coding Example (3)

符號	機率	Huffman code	碼長	總位元數
a_2	0.4	1	1	0.4
a_1	0.2	01	2	0.4
a_3	0.2	000	3	0.6
a_4	0.1	0010	4	0.4
a_5	0.1	0011	4	0.4

平均碼長 2.2 bits/symbol

Applications of Huffman Coding

Lossless Image Compression

Sena

Sensin

Earth

Omaha

- Four Test Images
- Graylevel is 0-255
- Image size is 256x256
- Uncompressed size is 65536 bytes

Lossless Image Compression

Compression using Huffman codes on pixel values.

Image Name	Bits/Pixel	Total Size (bytes)	Compression Ratio
Sena	6.9	56,533	1.16
Sensin	7.38	60,474	1.27
Earth	4.78	39,193	1.67
Omaha	7	57,356	1.14

Compression Ratio is

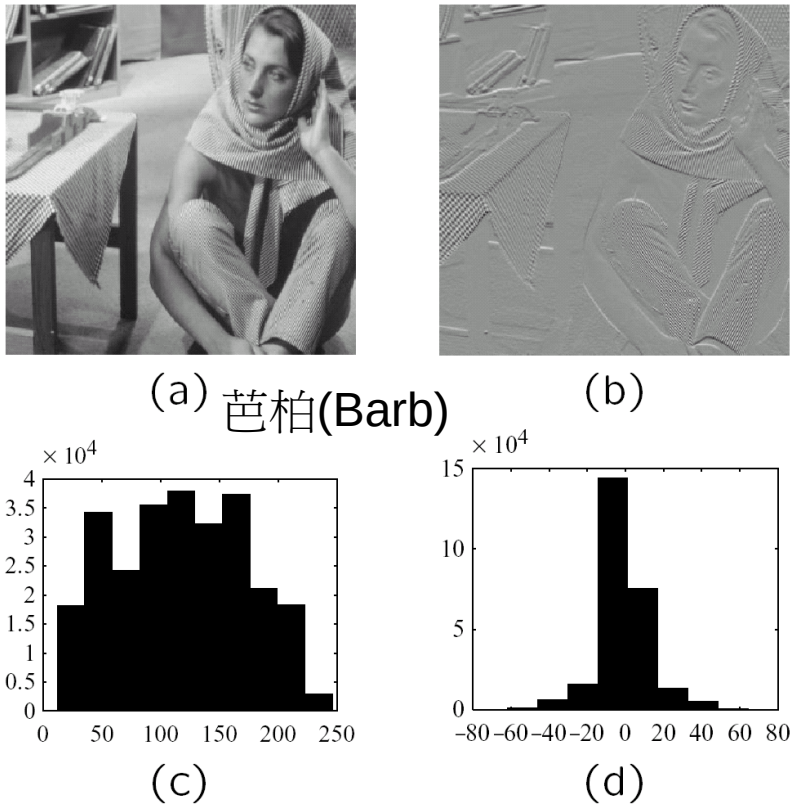
$$C_r = \frac{\text{Number of bits needed for the original signal}}{\text{Number of bits needed for the compressed signal}}$$

影像上的差動式編碼策略

- 給定一張影像$I(x, y)$，透過簡單的差動式運算我們就可以定義差動影像$d(x, y)$為：

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

- 由於一般影像I中存在空間冗餘性，故差異影像d中的色階分布範圍會較窄，也因此具有較小的熵



原始影像和動差影像的分布：

(a,b) 原來的灰階影像與其偏微分的影像

(c,d)原始影像和微分影像的色階分布圖

Lossless Image Compression

Compression using Huffman codes on pixel difference values.			
Image Name	Bits/Pixel	Total Size (bytes)	Compression Ratio
Sena	3.84	31,472	2.08
Sensin	4.63	37,880	1.73
Earth	3.92	32,136	2.04
Omaha	6.34	51,986	1.26

$$Let \quad \hat{x}_n = x_{n-1}$$

Pixel difference value is $\hat{x}_n - x_n$

Applications of Huffman Coding

Text compression

TABLE 3.20 **Probabilities of occurrence of the letters in the English alphabet in the U.S. Constitution.**

Letter	Probability	Letter	Probability
A	0.057305	N	0.056035
B	0.014876	O	0.058215
C	0.025775	P	0.021034
D	0.026811	Q	0.000973
E	0.112578	R	0.048819
F	0.022875	S	0.060289
G	0.009523	T	0.078085
H	0.042915	U	0.018474
I	0.053475	V	0.009882
J	0.002031	W	0.007576
K	0.001016	X	0.002264
L	0.031403	Y	0.011702
M	0.015892	Z	0.001502

Applications of Huffman Coding

Text compression

TABLE 3.21 **Probabilities of occurrence of the letters in the English alphabet in this chapter.**

Letter	Probability	Letter	Probability
A	0.049855	N	0.048039
B	0.016100	O	0.050642
C	0.025835	P	0.015007
D	0.030232	Q	0.001509
E	0.097434	R	0.040492
F	0.019754	S	0.042657
G	0.012053	T	0.061142
H	0.035723	U	0.015794
I	0.048783	V	0.004988
J	0.000394	W	0.012207
K	0.002450	X	0.003413
L	0.025835	Y	0.008466
M	0.016494	Z	0.001050