Entropy Coding

Huffman Coding

- Shannon code
- Huffman code
- Variations of Huffman code
- Tunstall code
- Colomb code

Shannon Coding

Shannon Coding

Symbols that occur more frequently will have shorter codewords (字碼) than symbols that occur less frequently

雪農編碼演算法：

步驟一： 計算所有符號的出現機率並將它們依機率的大小，由大到小排序。

步驟二： 將排序後的符號，依其機率值二分成兩個群組，使兩群組的符號機率總和值愈接近愈好。

步驟三： 分別將“0”和“1”指定給那二個群組。

步驟四： 重複步驟二和步驟三，直到每個群組只剩下一個符號才停止位元指定。

步驟五： 每個符號被指定的0和1位元集合即是代表該符號的字碼。

步驟六： 使用步驟五決定的符號字碼，一一編碼所有待壓縮符號。

解碼演算法則使用與編碼端相同的符號字碼表，並依相反的解碼動作來實現

Shannon Coding

Example :

Encode the sequence: B A C D E F G H A C D E F G C D D E E F D D E E

1. Determine the probability of each symbol (表4.1)

2. Sort, accumulate and partition the table (表4.2)

表 4.1

符 號	機 率
A	2/24=1/12
B	1/24
C	3/24=1/8
D	6/24=1/4
E	6/24=1/4
F	3/24=1/8
G	2/24=1/12
H	1/24

表 4.2

符號	機 率	累加機率	回合 1
D	1/4	1/4	0
E	1/4	2/4	0
C	1/8	5/8	1
F	1/8	6/8	1
A	1/12	10/12	1
G	1/12	11/12	1
B	1/24	23/24	1
H	1/24	24/24	1

Shannon Coding

表 4.3

符號	機　率	累加機率	回合 1	回合 2	回合 3	回合 4
D	1/4	1/4	0	0		
E	1/4	2/4	0	1		
C	1/8	5/8	1	0	0	
F	1/8	6/8	1	0	1	
A	1/12	10/12	1	1	0	0
G	1/12	11/12	1	1	0	1
B	1/24	23/24	1	1	1	0
H	1/24	24/24	1	1	1	1

編碼用字碼　D: 00　E: 01　C: 100　F: 101
A: 1100　G: 1101　B: 1110　H: 1111

Input Seq.: B A C D E F G H A C D E F G C D D E E F D D E E (24 symbols)

Fixed-length code needs 72 bits to encode the sequence

Shannon code needs 66 bits to encode the sequence

Huffman Coding

- A class assignment by David Huffman (1951) at MIT
- Based on two observations
  - In an optimal code, symbols that occur more frequently will have shorter codewords
  - In an optimal code, the 2 symbols that occur least frequently will have the same length

Huffman Coding

The algortihm：

步驟一： 計算每一個符號的出現機率，並將所有符號及其機率放入待處理符號集合R中，準備由下往上建立一棵編碼二元樹。

步驟二： 從待處理集合中**找出機率最小的兩個**符號做為二元樹的兩個子節點，並為這兩個節點建立一個父節點，此父節點機率為兩個子節點的機率和。再將這兩個子節點從R中移除，且把其父節點(含機率)加入R中。

步驟三： 重複步驟二直到待處理集合只剩下一個符號。

步驟四： 將建立完成的二元樹中任何兩兄弟節點的左接線標上0右接線標上1。樹中每個符號被指定之0與1的位元集合即是代表該符號的字碼。(此時，符號一定位於樹葉節點)

步驟五： 使用在步驟四決定的符號字碼，一一編碼所有待壓縮符號。

Huffman Coding

Example :

Encode the sequence: a e b a c d d a e a

Determine the probability of each symbol and construct the coding tree

P(a)=0.4,

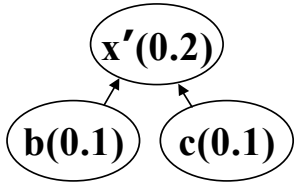
P(b)=0.1,

P(c)=0.1,

P(d)=0.2,

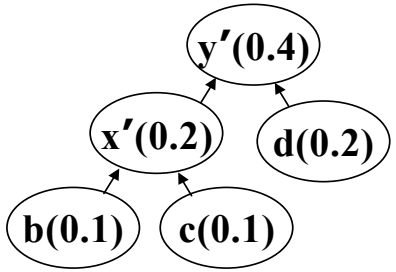
P(e)=0.2,

R={a, b, c, d, e}



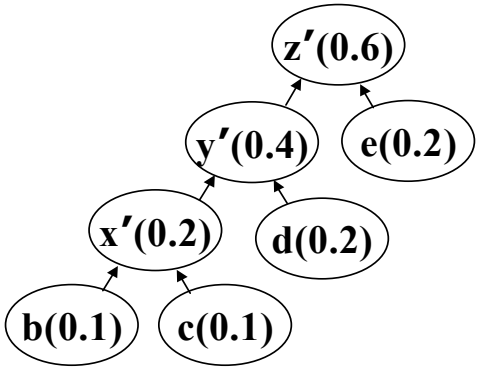
合併後R={a(0.4), x′(0.2), d(0.2), e(0.2)}

(a)



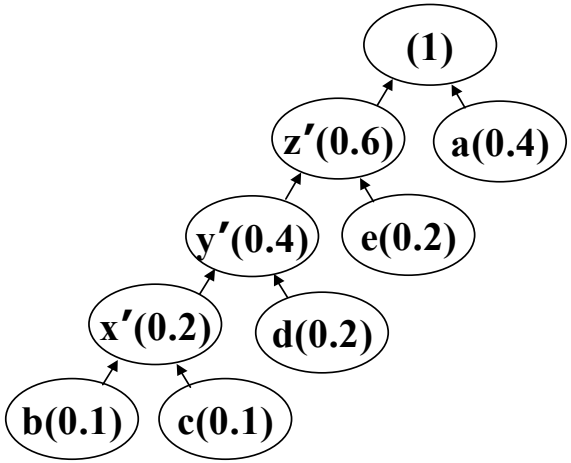
合併後R={a(0.4), y′(0.4), e(0.2)}

(b)



合併後R={a(0.4), z′(0.6)}

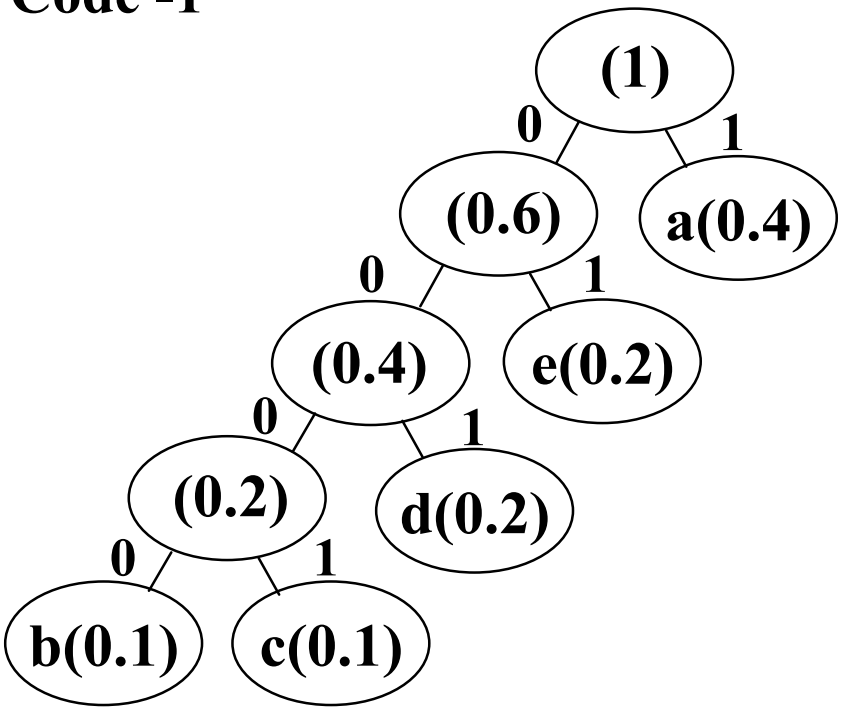
(c)



(d)

Huffman Coding

Code -1



符　號	字碼
a	1
e	01
d	001
b	0000
c	0001

the input sequence: a e b a c d d a e a

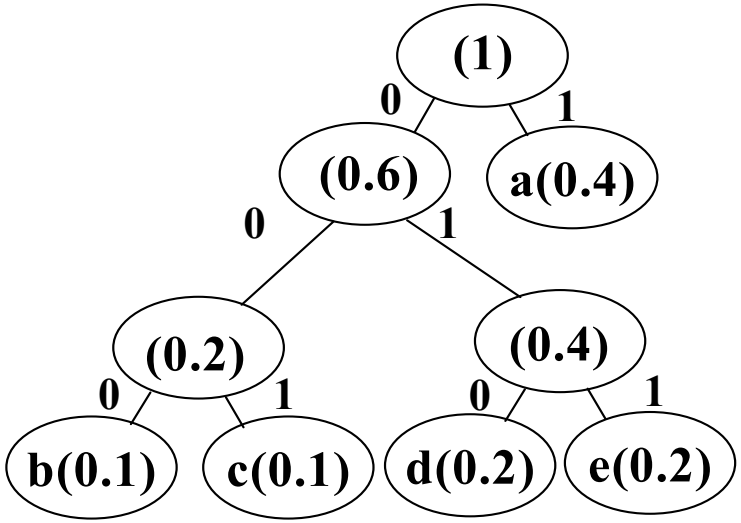
the encoded results: 1010000100010010011011

Huffman code needs 22 bits to encode the sequence

Fixed-length code needs 30 bits to encode the sequence – Why ?

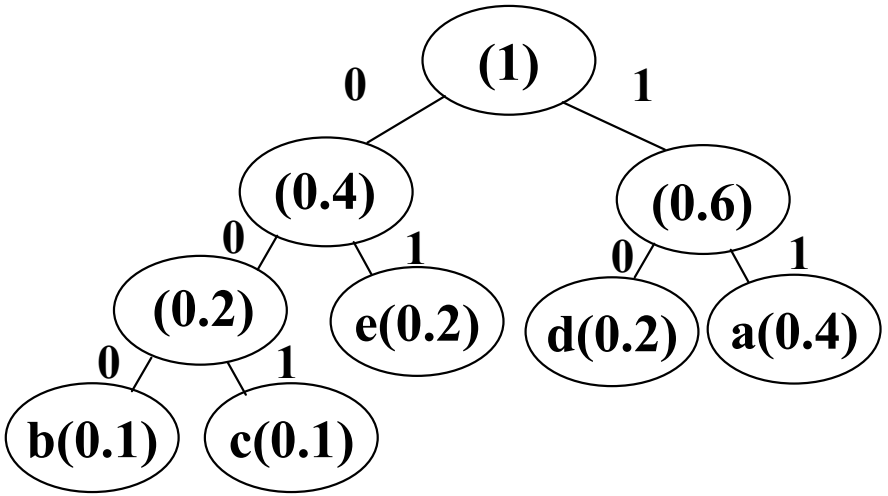
Huffman Coding

Code -2



符 號	字 碼
a	1
e	011
d	010
b	000
c	001

Code -3



符 號	字 碼
a	11
e	01
d	10
b	000
c	001

Both Code-2 and Code-3 need 22 bits to encode the input sequence

Huffman Decoding

The algorithm：(假設解碼端擁有編碼端所使用的編碼二元樹)

步驟一：將一個指標指到編碼二元樹的樹根節點。

步驟二：讀入一個待解碼位元，若此位元為**0**，則將指標移到目前所指節點的左子節點；若此位元為**1**，則將指標移到目前所指節點的右子節點。

步驟三：檢查目前指標所指的節點是否為樹葉節點：

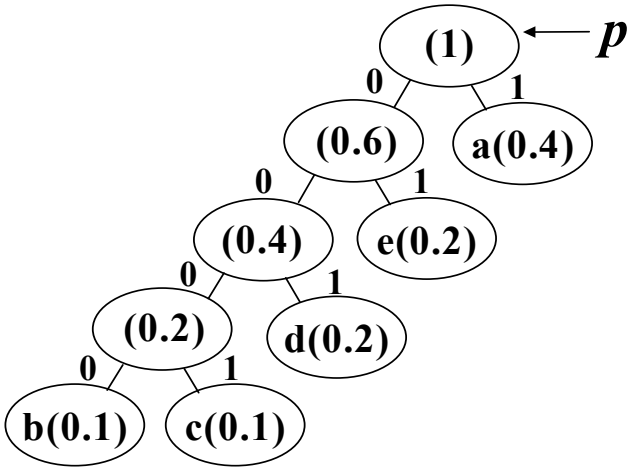
若是樹葉節點，則該節點所對應的符號就是解碼出來的符號，然後跳到步驟四。

若不是樹葉節點，則跳到步驟二，再讀入下一個位元。

步驟四：重複步驟一到三，直到所有待解碼位元都處理完畢。

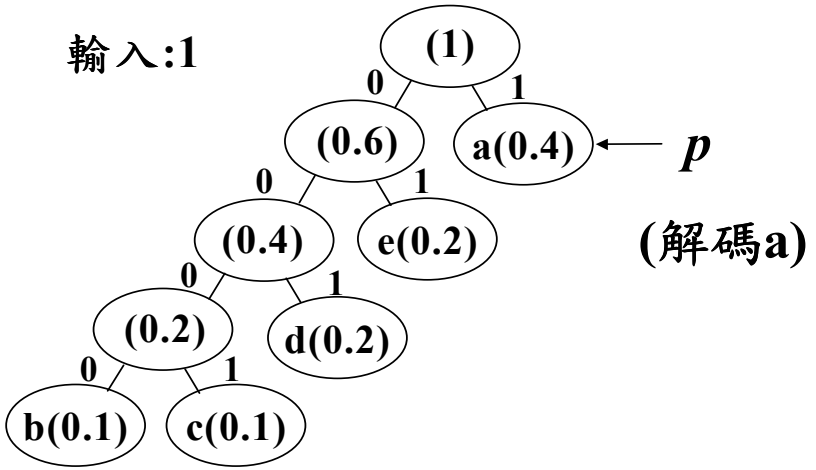
Huffman Coding

Sequence: 1010000100010010011011



(a)

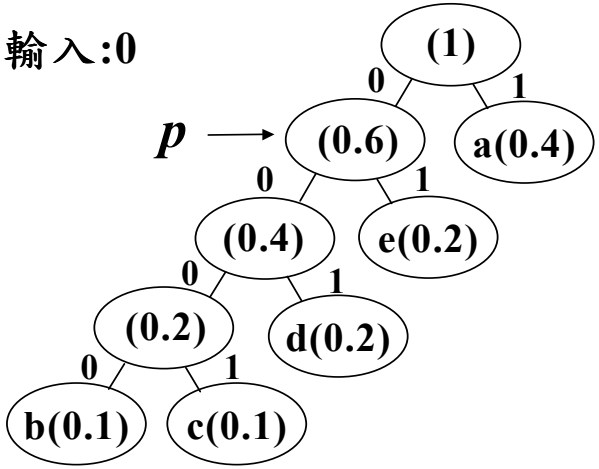
輸入:1



(解碼a)

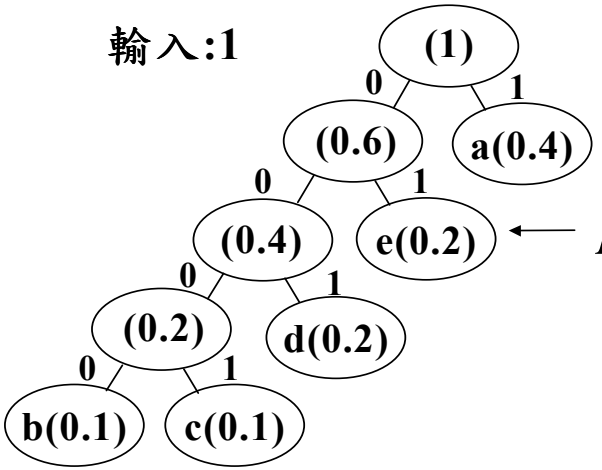
(b)

輸入:0



(c)

輸入:1



(解碼e)

(d)

Huffman Coding

Some issues for the implementation of Huffman coding:

1. How to find the probability of each symbol ?
   Static model:
   a. Prescanning: prescan the input sequence
      好處: better compression efficiency
      壞處: longer coding time, extra transmission time and storage
   b. Fixed probability (statistics): dictionary or encyclopedia
      好處: easy, less time and storage 壞處: worse compression efficiency
   Adaptive model:
      changeable probability (introduced later)
2. The compression efficiency will decrease largely when the number of symbols is small or their probabilities are highly skewed
3. The size of Huffman tree is limited by the available resources
4. A bit error will cause a series of errors

Minimum Variable-Length Huffman Coding

最小變動長度霍夫曼編碼演算法：

步驟一：計算每一個符號的出現機率，並將所有符號及其機率放入待處理符號集合*R*中，準備由下往上建立二元樹。

步驟二：從待處理集合中找出機率最小的兩個符號做為二元樹的兩個子節點，若機率最小的節點有超過兩個以上的選擇(因為這些節點都擁有相同的最小機率)，此時須注意--**儘量先選擇不是別人父節點的節點**(即不是合成出來的新節點)，或是**儘量先選擇那些擁有較少層子樹的節點**(是合成出來的新節點，但其下的子樹層數較少)。然後再為這兩個節點建立一個父節點，此父節點機率為兩個子節點的機率和。再將這兩個子節點從*R*中移除，且把其父節點(含機率)加入*R*中。

步驟三：重複步驟二直到待處理集合只剩下一個符號。

步驟四：將建立完成的二元樹中任何兩兄弟節點的接線分別標上0與1。

步驟五：使用在步驟四決定的符號字碼，一一編碼所有待壓縮符號。

Minimum Variable-Length Huffman Coding

Example :

Encode the sequence: a e b a c d d a e a

Determine the probability of each symbol and construct the coding tree

***P*(a)=0.4,**

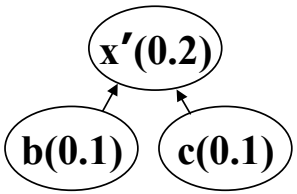
***P*(b)=0.1,**

***P*(c)=0.1,**

***P*(d)=0.2,**

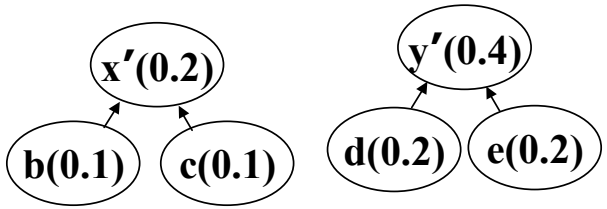
***P*(e)=0.2,**

***R*={a, b, c, d, e}**



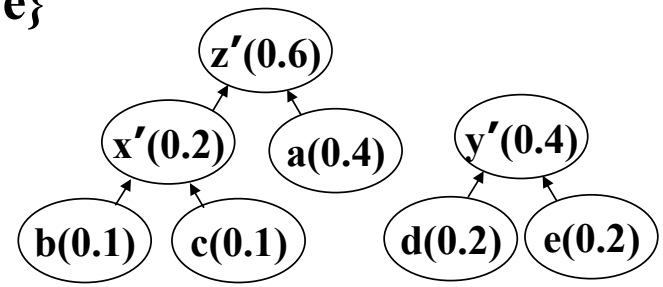
合併後*R*={a(0.4), x′(0.2), d(0.2), e(0.2)}

(a)



合併後*R*={a(0.4),x′(0.2), y ′(0.4)}

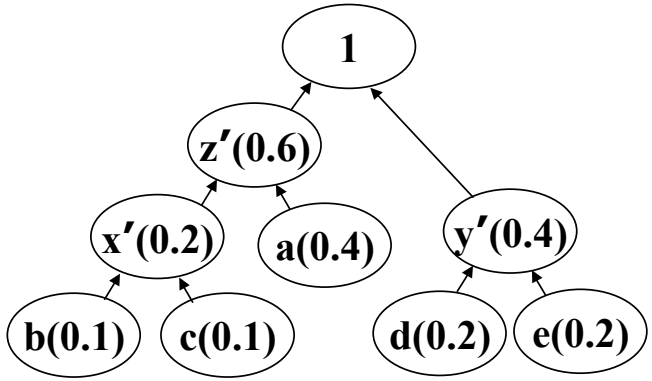
(b)



合併後*R*={y ′(0.4), z′(0.6)}

(c)

Code – 4 (MVLHC)

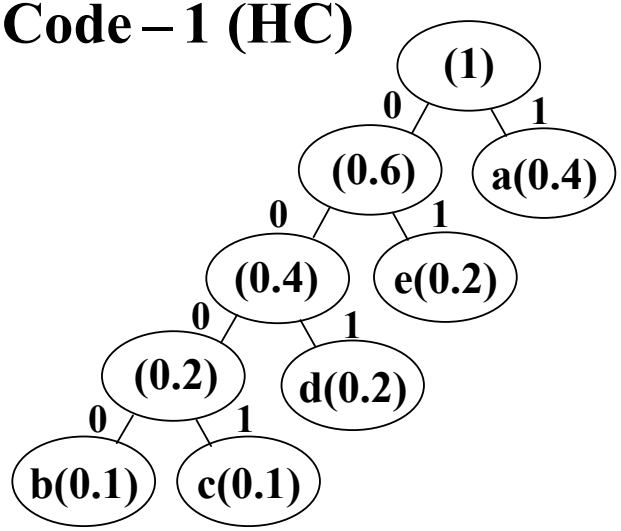


(d)

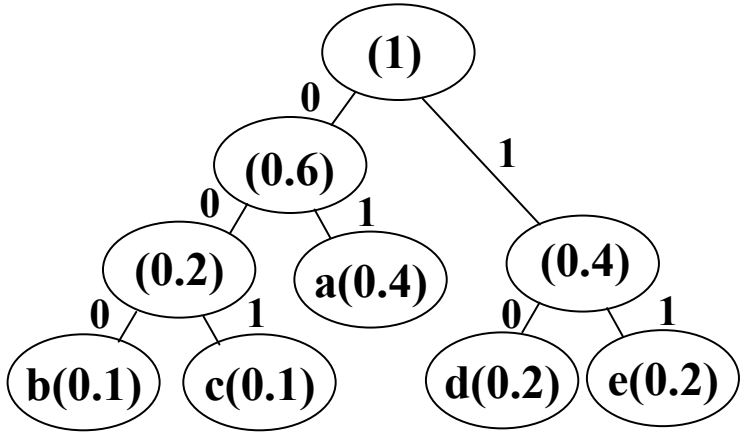
Minimum Variable-Length Huffman Coding

最小變動長度霍夫曼編碼法所編出來的碼，其實只能算是霍夫曼編碼法所編出的許多種可能編碼方式中的一種，故其編碼效率和霍夫曼編碼一樣。但是，

Code – 1 (HC)



Code – 4 (MVLHC)



In average, the performances of the two Huffman codes (Code-1 and Code-4) are the same, but when buffer transmission is considered:

Transmit 1000 b = 4000 bits (Code-1) =3000bits (Code-4)

Transmit 1000 a = 1000 bits (Code-1) =2000bits (Code-4)

If transmission capacity is 2500 bits/per second, buffer size ?

Code-4 (MVLHC) needs less buffer size

Extended(Vector) Huffman codes

- When the alphabet is small and the probability of occurrence of the different symbol is skewed
  - H(x) « 1 bit/symbol
  - Huffman code is very inefficient
- Remedy:
  - Combine *m* successive symbols to a new “block symbol”
  - Huffman code for block symbols
  - Disadvantage: exponentially growing alphabet size

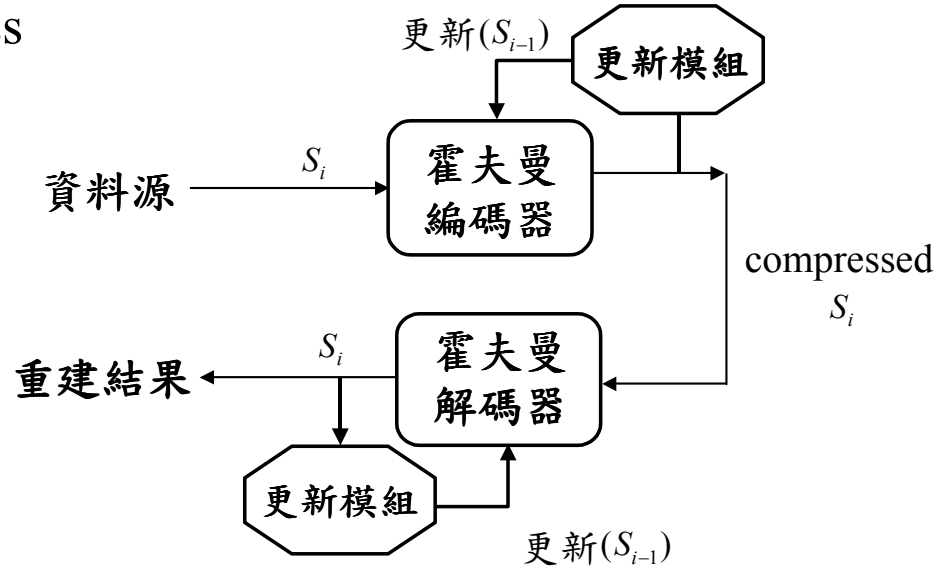
Adaptive Huffman Coding

Huffman coding:

The coding tree and codewords won't change through the whole coding process

Adaptive Huffman coding:

- To achieve better probability estimation, we use adaptive model to tune the probability of each symbol dynamically
- Better probability estimation can guarantee better compression efficiency
- The coding tree and codewords change dynamically through the whole coding process



Adaptive Huffman Coding

基本的適應性霍夫曼編碼演算法：

步驟一：計算每一個符號的出現機率，並將所有符號及其機率放入待處理符號集合R中，準備由下往上建立一棵編碼二元樹。

步驟二：從待處理集合中找出機率最小的兩個符號做為二元樹的兩個子節點，並為這兩個節點建立一個父節點，此父節點機率為這兩個子節點的機率和。並將這兩個子節點從R中移除，且把其父節點(含機率)加入R中。

步驟三：重複步驟二直到待處理集合只剩一個符號。

步驟四：將建立完成的二元樹中任何兩個兄弟節點的接線分別標上0與1。樹中每個符號被指定的0與1的位元集合即是代表該符號的字碼。

步驟五：使用在步驟四決定的符號字碼，編碼目前待編碼符號(S_i)。

步驟六：**更新模組(update module)使用剛剛編碼過的符號(S_i)來更新所有符號的出現機率。令i=i+1，再重複步驟一到六，每編碼完一個符號，計算新的符號機率，再重建一棵新的編碼樹，然後編碼下一個輸入符號，重複此動作一直到檔案結束。**

Adaptive Huffman Coding

基本的適應性霍夫曼解碼演算法：

(假設解碼端擁有編碼端所使用的初始編碼二元樹)

步驟一： 將一個指標指到編碼二元樹的樹根節點。

步驟二： 讀入一個待解碼位元，若此位元為0，則將指標移到目前所指節點的　左子節點；若此位元為1，則將指標移到目前所指節點的右子節點。

步驟三： 檢查目前指標所指的節點是否為樹葉節點：

若是樹葉節點，則該節點所對應的符號就是解碼出來的符號，然後跳到步驟四。

若不是樹葉節點，則跳到步驟二，再讀入下一個位元。

步驟四： 更新模組(update module)使用剛剛解碼出的符號(S_i)來更新所有符號的出現機率並重建一棵新的解碼樹(如編碼演算法中的步驟一到四)。然後再重複解碼步驟一到四，解碼下一個位元，直到所有待解碼位元都處理完畢。

Adaptive Huffman Coding

更新模組(update module)可使用下述方式來更新符號的機率：

編碼前(或解碼前)，每一個符號S_i的出現機率可用下式計算：

$$P(S_i) = count(S_i) / count(total),\ i = 1,2,3,...,n$$

其中輸入符號集共含有n個不同的符號，$count(S_i)$代表符號S_i到目前為止的出現次數，$count(total)$是到目前為止符號的總出現次數。

若目前輸入編碼(或解碼出)的符號是x，霍夫曼編碼後(或解碼後)更新模組需更新所有的符號出現機率如下所示：

$$P(S_i) = count(S_i)/(count(total)+1), i = 1,2,3,..,n \text{ 且 } s_i \neq x$$

$$P(S_i) = (count(S_i)+1)/(count(total)+1), \text{當 } s_i = x$$

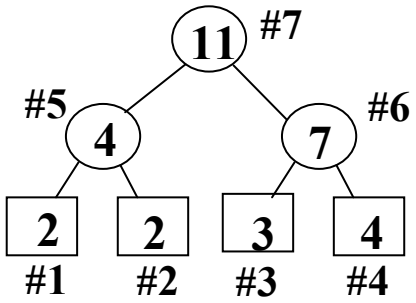
符號x的出現次數要加1次，其他符號的出現次數則不變。

適應性霍夫曼編碼的過程中，有一項極為嚴重的缺點：**那就是每編碼一個符號後，我們必須重複地更新所有符號的出現機率並重建編碼樹---這是相當耗時且可怕的一件事** 。

Adaptive Huffman Coding

A more practical approach for the implementation of AHC:

1. Add a **node number** and a **weight** to each node
   - the weight of an **external node** (symbol) is its **occurring times**
   - the weight of an **internal node** is **the sum of its offspring**
2. Guarantee the sibling property (兄弟性質)
   - the nodes y_{2j-1} and y_{2j} are offsprings of the same parent node
   - the node number of an internal node is **greater than** its offspring
   - x_j is the weight of node y_j, thus $x_1 \le x_2 \le .... \le x_{2n-1}$

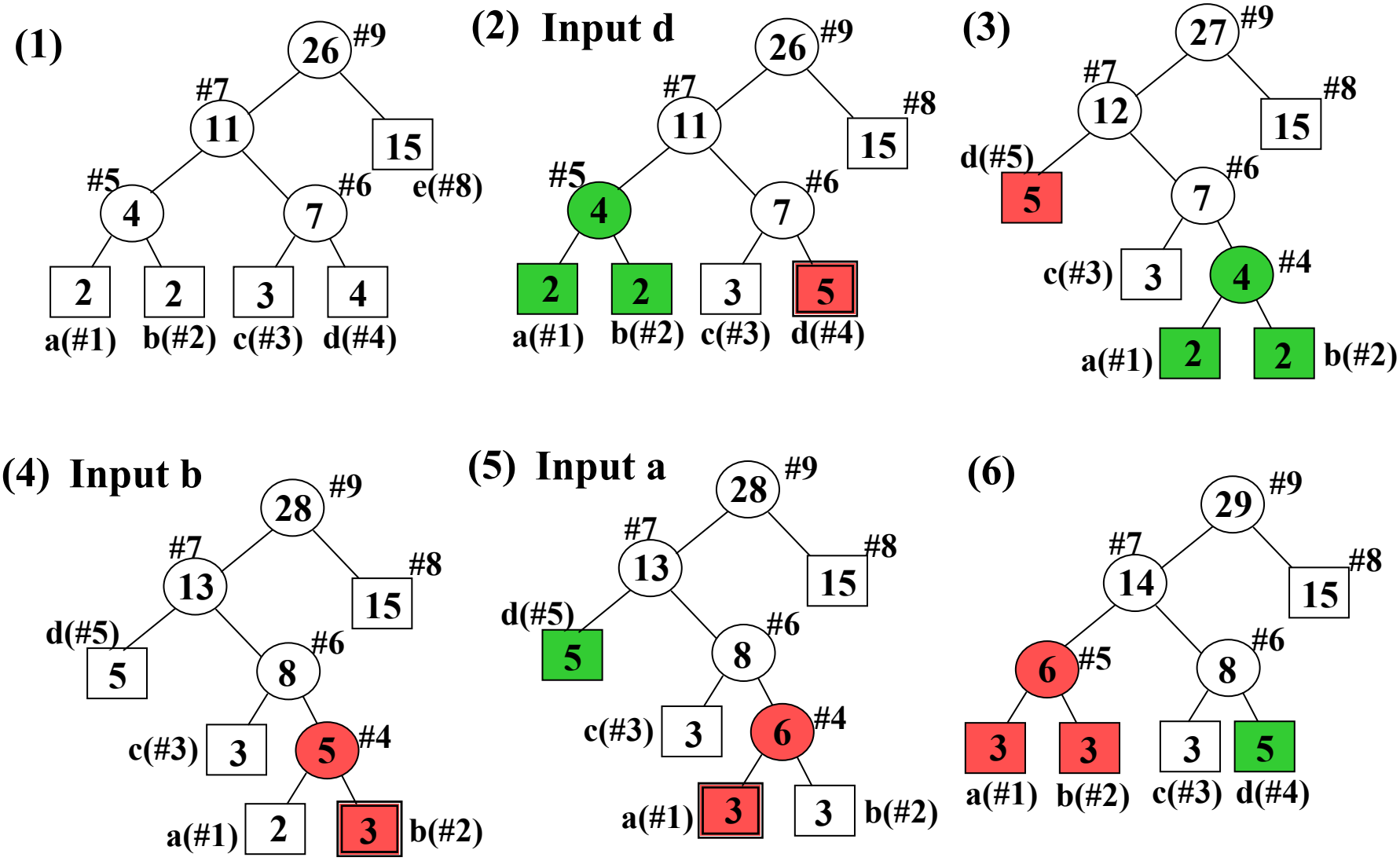


n=4 (leaves),
$2n$-1=7 nodes,
x_1=2, x_2=2,…, x_6=7, x_7=11

When updating the adaptive Huffman tree:

Guarantee the sibling property through out the whole process

Adaptive Huffman Coding



Example for Adaptive Huffman Code

Example :

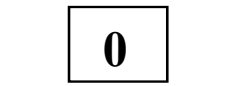
Encode the sequence: x y z x z

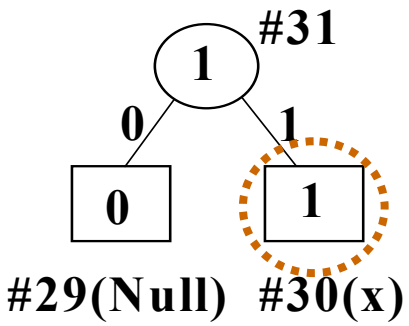
假設要編碼的序列只包含26個小寫英文字母，因此需使用5個位元來表示26種組合。所以首先建立一個符號代碼對照表，讓編碼端與解碼端同時使用:

符號	字碼(5bits)
a	00000 (0)
b	00001 (1)
c	00010 (2)
. .	
x	11000 (23)
y	11001 (24)
z	11010 (25)

Input: x
Output: 11000

Input: y
Output: 011001





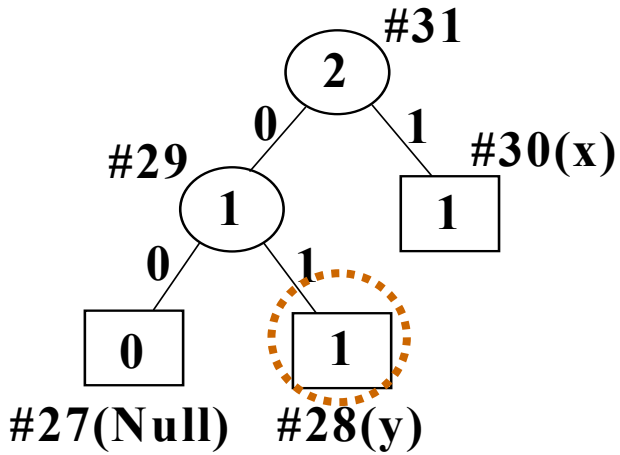


圖 4.15(a)

圖 4.15(b)

圖 4.15(c)

Example for Adaptive Huffman Code

Input: z
Output: 0011010

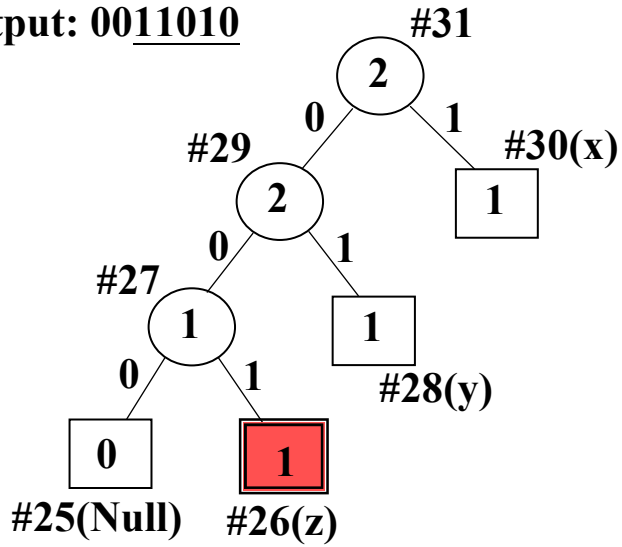


圖4.15(d)

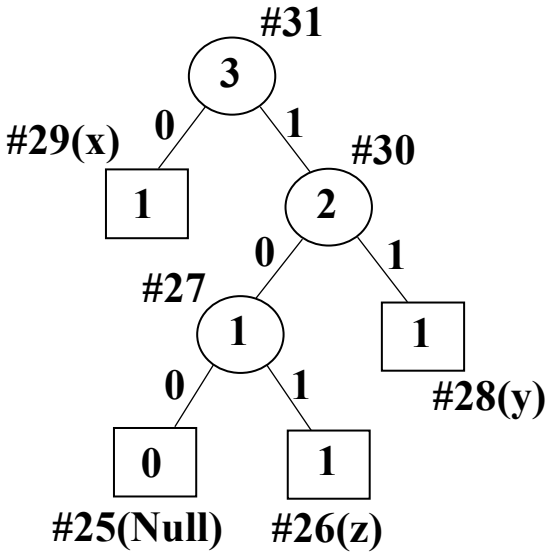


圖4.15(e)

Input: x
Output: 0

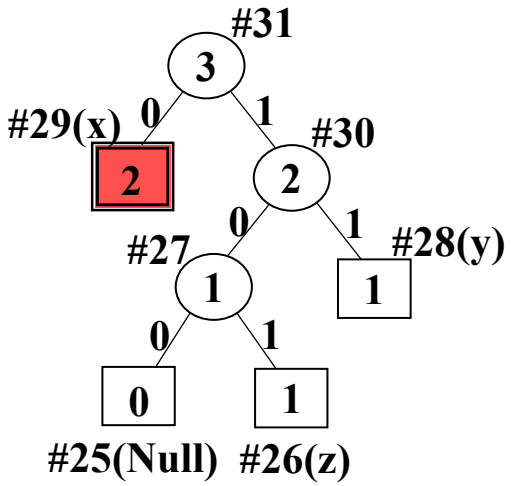


圖4.15(f)

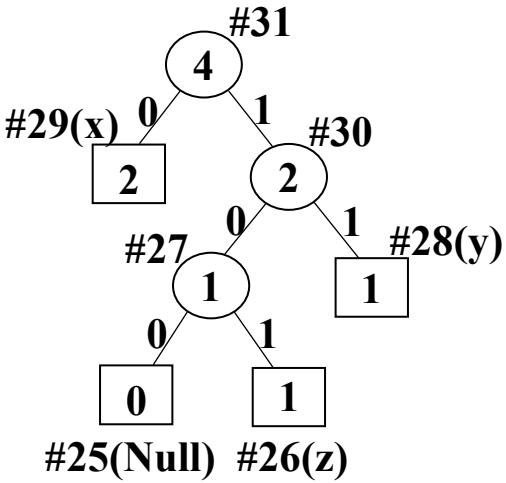


圖4.15(g)

Example for Adaptive Huffman Code

Input: z
Output: 101

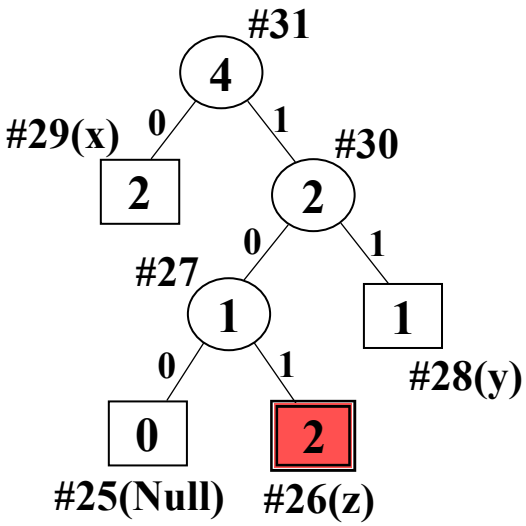


圖4.15(h)

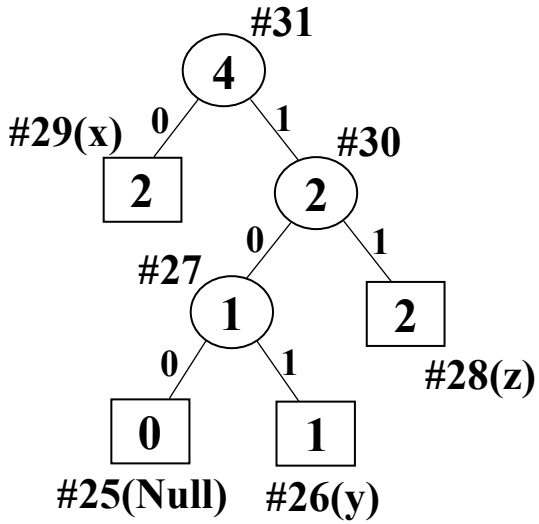
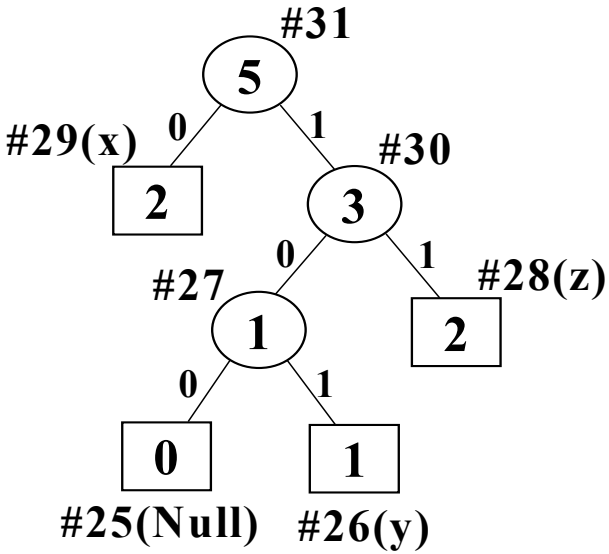


圖4.15(i)

Input data: xyzxz
Output: 11000011001
00110100101



See page 75, textbook

Tunstall Coding(湯斯多編碼法)

Two special properties for Tunstall code:

a. All codewords are of equal length.

b. Each codeword represents a different number of symbols

Example 4.6: (page 85, textbook)

Encode the sequence: **z x y x y y y z x y** with 3-bit Tunstall code

The probabilities of the symbols are: $K=H=\{x,y,z\}$ $P(x)=0.3$, $P(y)=0.5$, $P(z)=0.2$

a. Remove the symbol that has the highest probability in H, and concatenate the symbol with every symbol in K

$H=\{x(0.3), z(0.2), yx(0.15), yy(0.25), yz(0.1)\}$

where $P(yx)=P(y)\times P(x)$，$P(yy)=P(y)\times P(y)$，$P(yz)=P(y)\times P(z)$

b. Repeat procedure a

$H=\{z(0.2), yx(0.15), yy(0.25), yz(0.1), xx(0.09), xy(0.15), xz(0.06)\}$

Because the number of symbols in set H is 7, we end the process

Tunstall Coding

Example 4.6:

字　串	機　率	字　碼
z	0.2	000
yx	0.15	001
yy	0.25	010
yz	0.1	011
xx	0.09	100
xy	0.15	101
xz	0.06	110

湯斯多編碼法為什麼要將出現機率較高者與其他字母結合呢？其實它利用的也是**將出現機率較高的字元用較短的字碼來表示，出現機率較低的字元用較長的字碼來表示**的原理。Why？

當然湯斯多編碼法的編碼效果不會比霍夫曼編碼法來的好，但是它有一個很好的優點--**當湯斯多編碼結果在傳輸過程發生一個位元的傳輸錯誤時，其解碼端並不會解出一連串的錯誤(即錯誤不會往後傳遞)**。大部份的變動長度編碼都有一個嚴重的問題，當編碼結果傳輸至解碼端時，若發生一個位元的傳輸錯誤時，可能會造成之後一連串的解碼錯誤，而湯斯多編碼法卻克服了這個問題，目前的字碼發生傳輸錯誤，只有此字碼會解碼錯誤，不會影響後面字碼的解碼。

Advantages of Tunstall Code

Example :

P(a)=0.4，P(b)=0.3，P(c)=0.2，P(d)=0.1

Encode the sequence: c b d c with Huffman code

字 元	字 碼
a	1
b	01
c	001
d	000

outputs: 00101000001

one-bit error: 00100000001

decompression: cddb

Encode the sequence: c b d c with 3-bit Tunstall code

字 元	機 率	字 碼
d	0.1	000
c	0.2	001
b	0.3	010
ad	0.04	011
ac	0.08	100
ab	0.12	101
aa	0.16	110

outputs: 001010000001

one-bit error: 00100000001

decompression: cddc

Adv: errors in codewords do not propagate

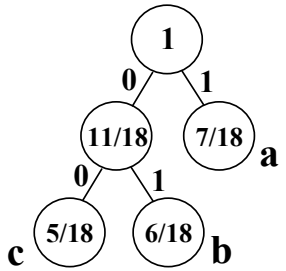
(In VLC, a bit error will cause a series of errors)

n-order Markov-Huffman Coding

一個好的模式，可達到較好的編碼效率。若將馬可夫模式與霍夫曼編碼相結合，藉由馬可夫來增進機率估計的準確性，應能提昇霍夫曼編碼的壓縮效率。

input sequence：a b a b c a b b c b a c c a a a b c (prescanning)

zero-order Markov-Huffman Code ⇒

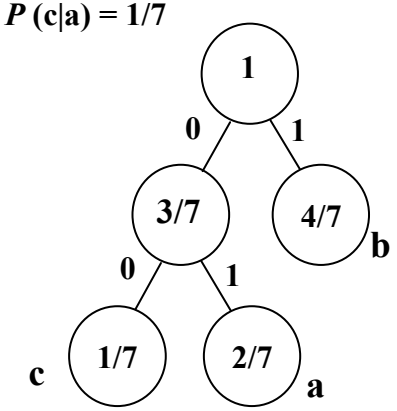


a:1 b:01 c:00

first-order Markov-Huffman Coding (three symbols in the alphabet)

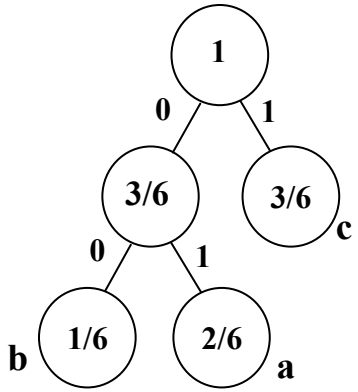
狀態a的編碼樹: *P*(?|a)

***P*(a|a) = 2/7, *P*(b|a) = 4/7, *P*(c|a) = 1/7**



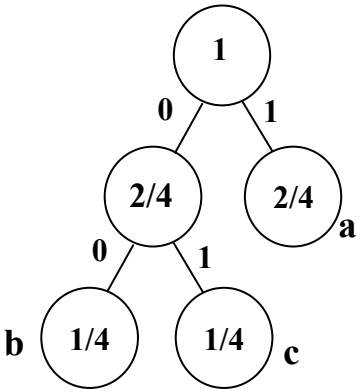
狀態b的編碼樹: *P*(?|b)

***P*(a|b) = 2/6, *P*(b|b) = 1/6, *P*(c|b) = 3/6**



狀態c的編碼樹: *P*(?|c)

***P*(a|c) = 2/4, *P*(b|c) = 1/4, *P*(c|c) = 1/4**



★ **How to determine the prob of *P* (a|a), *P* (a|b), … ?**

n-order Markov-Huffman Coding

sequence：a b a b c a b b c b a c c a a a b c (1-order Markov-Huffman Coding)

已編碼符號| 待編碼符號（假如預設狀態為b）

序列: |ababca

使用狀態b 的樹

a ➡ 01

輸 出: 01

(a)

序列: a|babca

使用狀態a 的樹

b ➡ 1

輸 出: 011

(b)

序列: ab|abca

使用狀態b 的樹

a ➡ 01

輸 出: 01101

(c)

序列: aba|bca

使用狀態a 的樹

b ➡ 1

輸 出: 011011

(d)

序列: abab|ca

使用狀態b 的樹

c ➡ 1

輸 出: 0110111

(e)

序列: ababc|a

使用狀態c 的樹

a ➡ 1

輸 出: 01101111

(f)

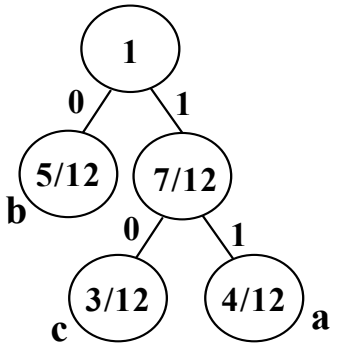
2-order Markov-Huffman Code: 9 coding trees (3 symbols in the alphabet)

***n*-order Markov-Huffman Code: 3^n coding trees (3 symbols in the alphabet)**

n-order Markov-Adaptive-Huffman Coding

適應性霍夫曼編碼與非適應性霍夫曼編碼最大的不同是--適應性霍夫曼編碼法每編碼一個符號，就必須更新霍夫曼樹一次

假設編碼到現在的適應性霍夫曼編碼樹如右圖



input sequence：c a a b c b

zero-order Markov-Adaptive-Huffman Coding

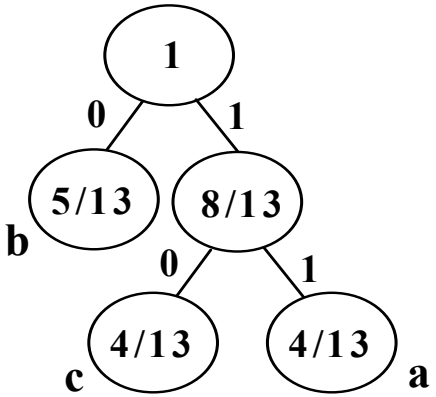
已編碼符號 | 待編碼符號

序列： |caabcb

c ⇨ 10

輸出：10

編碼樹更新為：



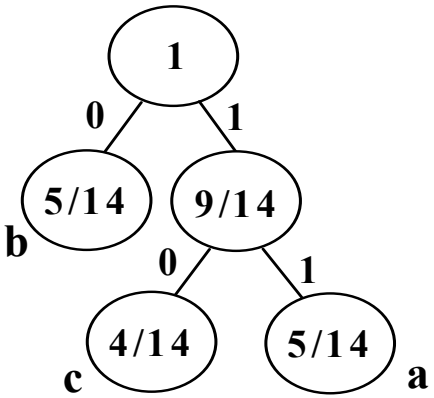
(a)

序列： c|aabcb

a ⇨ 11

輸出：1011

編碼樹更新為：



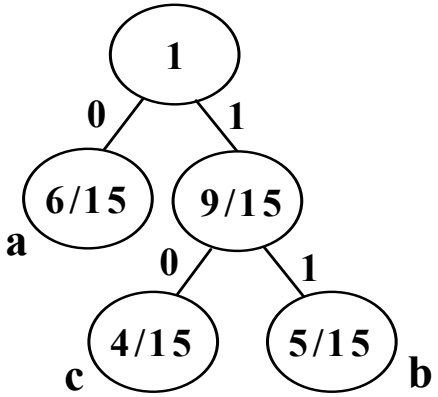
(b)

序列： ca|abcb

a ⇨ 11

輸出：101111

編碼樹更新為：



(c)

n-order Markov-Adaptive-Huffman Coding

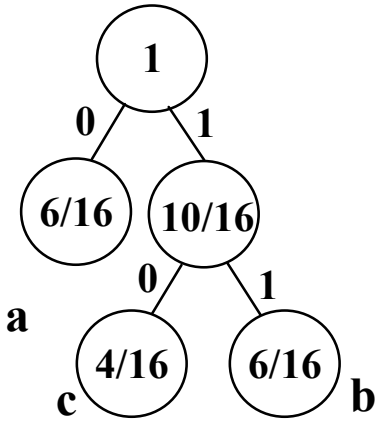
zero-order Markov-Adaptive-Huffman Coding

序列: caa|bcb

b ⇨ 11

輸 出: 10111111

編碼樹更新為:



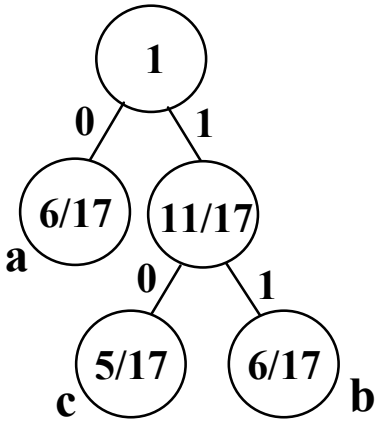
(d)

序列: caab|cb

c ⇨ 10

輸 出: 1011111110

編碼樹更新為:



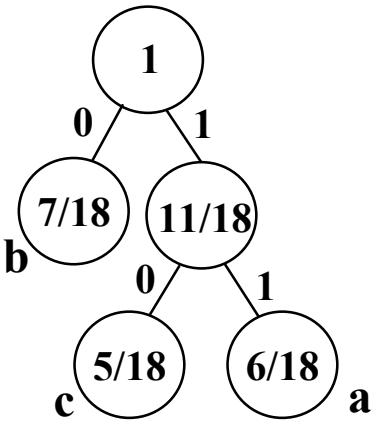
(e)

序列: caabc|b

b ⇨ 11

輸 出: 101111111011

編碼樹更新為:



(f)

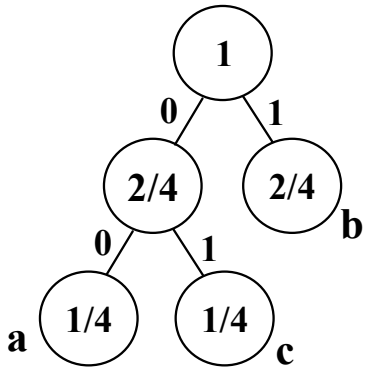
n-order Markov-Adaptive-Huffman Coding

first-order Markov-Adaptive-Huffman Coding (3 symbols in the alphabet)

1階馬可夫-適應性霍夫曼編碼法中，每編碼一個字元，就必須更新三棵霍夫曼樹中的某一棵。

狀態a的編碼樹: $P(?|a)$

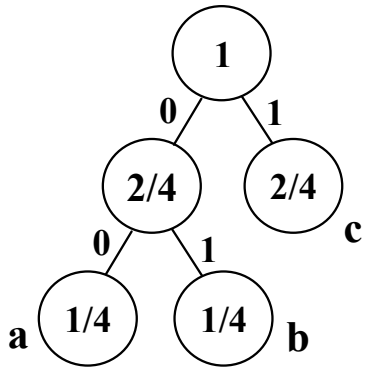
$P(a|a) = 1/4$, $P(b|a) = 2/4$,
$P(c|a) = 1/4$



(a)

狀態b的編碼樹: $P(?|b)$

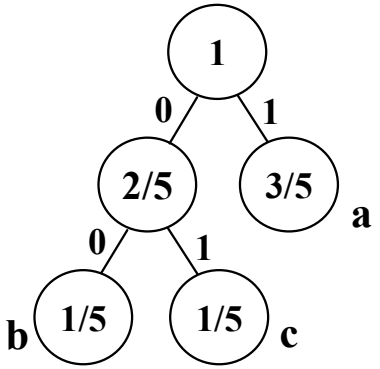
$P(a|b) = 1/4$, $P(b|b) = 1/4$,
$P(c|b) = 2/4$



(b)

狀態c的編碼樹: $P(?|c)$

$P(a|c) = 3/5$, $P(b|c) = 1/5$,
$P(c|c) = 1/5$

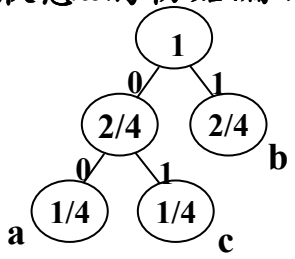


(c)

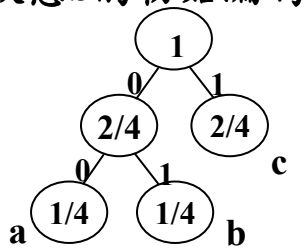
初始的一階馬可夫-適應性霍夫曼編碼樹集

n-order Markov-Adaptive-Huffman Coding

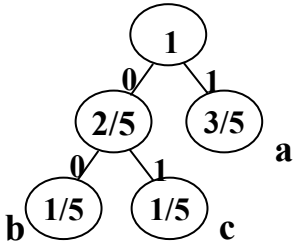
狀態a的初始編碼樹



狀態b的初始編碼樹



狀態c的初始編碼樹



first-order Markov-Adaptive-Huffman Coding

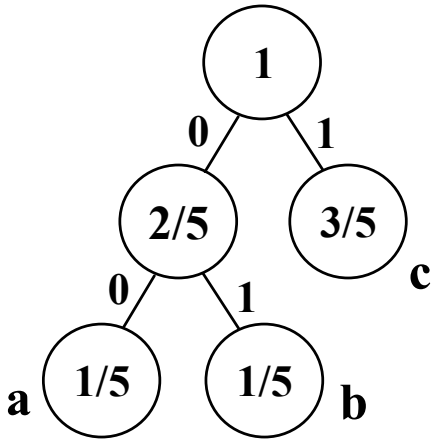
序列: |caabcb

使用編碼樹b

c ⇨ 1

輸 出: 1

編碼樹b更新為:



(a)

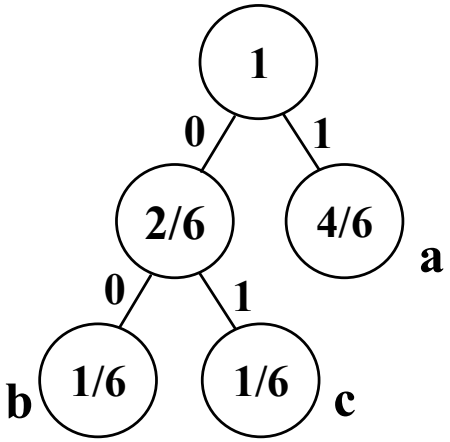
序列: c|aabcb

使用編碼樹c

a ⇨ 1

輸 出: 11

編碼樹c更新為:



(b)

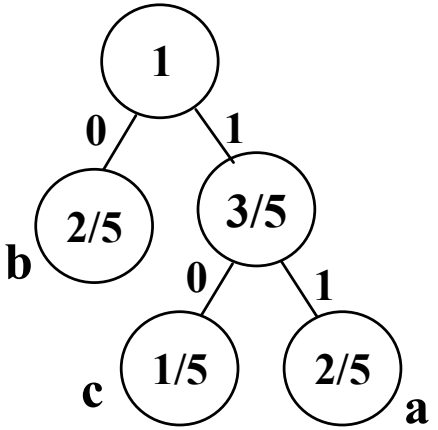
序列: ca|abcb

使用編碼樹a

a ⇨ 00

輸 出: 1100

編碼樹a更新為:



(c)

n-order Markov-Adaptive-Huffman Coding

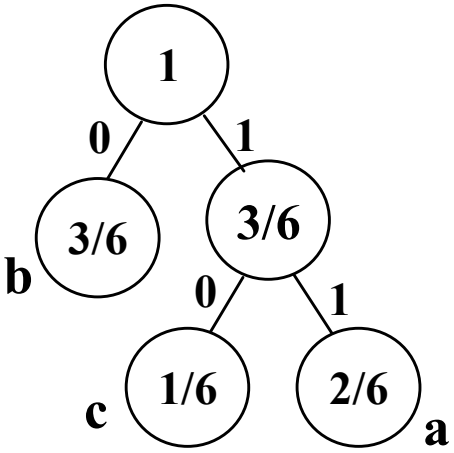
序列: caa|bcb

使用編碼樹a

b ⇨ 0

輸 出: 11000

編碼樹a更新為:



(d)

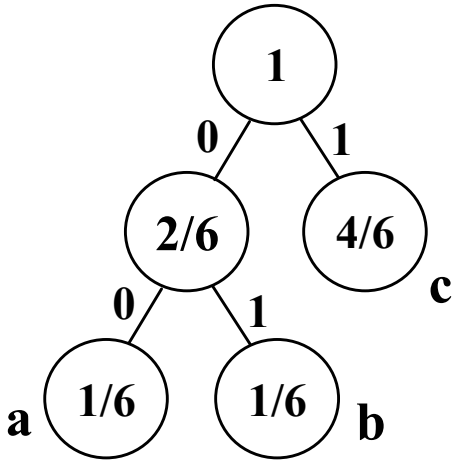
序列: caab|cb

使用編碼樹b

c ⇨ 1

輸 出: 110001

編碼樹b更新為:



(e)

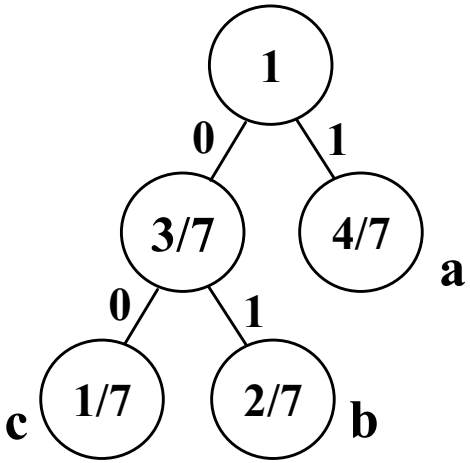
序列: caabc|b

使用編碼樹c

b ⇨ 00

輸 出: 11000100

編碼樹c更新為:



(f)

2-order Markov-Adaptive-Huffman Coding: 9 coding trees (3 symbols)

***n*-order Markov-Adaptive-Huffman Coding: 3^n coding trees (3 symbols)**

k^n coding trees (k symbols)

Colomb code

- Unary code for integer *n* is simply *n* 1s followed by a 0 (反之亦然)
- Combination of two parts
  - a unary code and a different code
  - A family of codes parameterized by an integer $m > 0$, let $n > 0$ and $q = \left\lfloor \frac{n}{m} \right\rfloor$, $r = n - qm$
  - quotient *q* is represented in unary code
  - 14= 2*5 +4

    110 111