Disadvantages of Huffman code

- For a source alphabet with small alphabets, such as binary sources, or with highly skewed probabilities (i.e. $P_{max} \uparrow$)
- the modeling & coding aspects of lossless compression are to be kept separate (why?)

The disadvantages of Huffman coding :

1. $H(s) \le \bar{l} < H(s) + P_{max} + 0.086 \quad for \quad P_{max} < 0.5$

   $H(s) \le \bar{l} < H(s) + P_{max} \quad for \quad P_{max} \ge 0.5$

   $\therefore P_{max} \uparrow$ inefficient

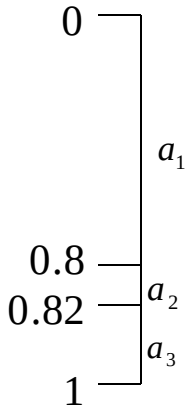
2. extended Huffman code: It does not always works efficiently
   - large storage : codebook increase exponentially
   - time-consuming decoding procedure

※ While a sequence of length *m* to be encoded, a unique arithmetic code (a tag) just be generated for the sequence without the need for generating codewords for all sequences of length *m* in advance.

Arithmetic Coding

Example :

Letters	$P(a_i)$	$F_X(i)$	Range
a_1	0.8	0.8	[0,0.8)
a_2	0.02	0.82	[0.8,0.82)
a_3	0.18	1	[0.82,1)



$$F_X(i) = \sum_{k=1}^{i} P(a_k)$$

Where *X* is the random variable with probability density function $P(X=i) = P(a_i)$, and cumulative density function $F_X(i)$.

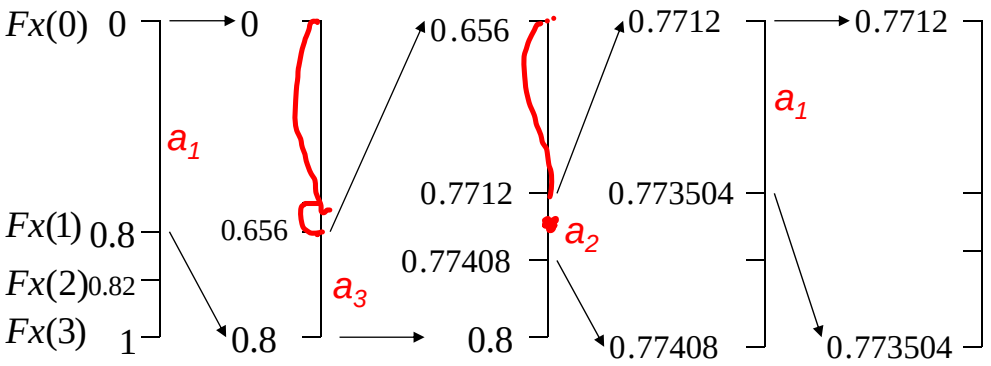
- associate the subinterval $[F_X(i\text{-}1), F_X(i))$ with the symbol a_i.
- $$\bar{T}_X(a_i) = \sum_{k=1}^{i-1} P(X=k) + \frac{1}{2}P(X=i)$$
  $$= F_X(i-1) + \frac{1}{2}P(X=i)$$

Arithmetic Encoding (1)

Input sequence : X = ($x_1, x_2, \ldots\ldots x_n$) = (1 3 2 1)

i.e. $a_1\ a_3\ a_2\ a_1$

$$\bar{T}_X(a_1a_3a_2a_1) = \frac{l^{(4)}+u^{(4)}}{2} = 0.772352 = (0.11000101\ldots)_2$$



$(0.11000101)_2 = 0.76953125$

$(0.11000110)_2 = 0.7734375$

$l^{(0)}=0$ $\quad l^{(1)}=0$ $\quad l^{(2)}=0.656$ $\quad l^{(3)}=0.7712$ $\quad l^{(4)}=0.7712$

$u^{(0)}=1$ $\quad u^{(1)}=0.8$ $\quad u^{(2)}=0.8$ $\quad u^{(3)}=0.77408$ $\quad u^{(4)}=0.773504$

Arithmetic Encoding (2)

$$\begin{cases} l^{(n)} = l^{(n-1)} + (u^{(n-1)} - l^{(n-1)})F_X(x_n - 1) \\ u^{(n)} = l^{(n-1)} + (u^{(n-1)} - l^{(n-1)})F_X(x_n) \end{cases}$$

$$\bar{T}_X(x) = F_X(x-1) + \frac{1}{2}P(x)$$

x : a sequence

$x-1$: the first sequence precedes sequence x

$$\begin{cases} l^{(2)} = l^{(1)} + (u^{(1)} - l^{(1)})F_X(x_2 - 1) = 0 + (0.8-0)F_X(2) \\ u^{(2)} = l^{(1)} + (u^{(1)} - l^{(1)})F_X(x_2) = 0 + (0.8-0)F_X(3) \end{cases}$$

$$\begin{cases} l^{(3)} = l^{(2)} + (u^{(2)} - l^{(2)})F_X(x_3 - 1) = 0.656 + (0.8-0.656)Fx(1) \\ u^{(3)} = l^{(2)} + (u^{(2)} - l^{(2)})F_X(x_3) = 0.656 + (0.8-0.656)Fx(2) \end{cases}$$

$$\bar{T}_X(a_1a_3) = P(a_1a_1) + P(a_1a_2) + \frac{1}{2}P(a_1a_3)$$

$$= 0.8\times0.8 + 0.8\times0.02 + \frac{1}{2}\times0.8\times0.18$$

$$= 0.8\times0.82 + \frac{1}{2}\times0.8\times0.18$$

Q: $\bar{T}_X(a_1a_3a_2)$

$= P(a_1a_1a_1) + P(a_1a_1a_2) + P(a_1a_1a_3)$ $\quad P(a_1a_1)$

$P(a_1a_2a_1) + P(a_1a_2a_2) + P(a_1a_2a_3)$

$P(a_1a_3a_1) + \frac{1}{2}P(a_1a_3a_2)$ $\quad P(a_1a_2)$

In fact, $\bar{T}_X(x) \in [\,F_X(x-1),\ F_X(x)\,)$

Arithmetic Decoding (1)

$tag = 0.772352$

$$\begin{cases} l^{(0)} = 0 \\ u^{(0)} = 1 \end{cases} \Rightarrow \begin{cases} F_X(0) = 0.00 \\ F_X(1) = 0.80 \\ F_X(2) = 0.82 \\ F_X(3) = 1.00 \end{cases}$$

$k=1 \quad t^* = 0.772352$

$\therefore x_1 = 1 \quad l^{(1)} = 0 \quad u^{(1)} = 0.8$

$\Downarrow$

$k=2 \quad t^* = \dfrac{0.772352 - 0}{0.8 - 0} = 0.96544$

$\therefore x_2 = 3 \quad l^{(2)} = 0.656 \quad u^{(2)} = 0.8$

$\Downarrow$

$k=3 \quad t^* = \dfrac{0.772352 - 0.656}{0.8 - 0.656} = 0.808$

$\therefore x_3 = 2 \quad l^{(3)} = 0.7712 \quad u^{(3)} = 0.77408$

$\Downarrow$

$k=4 \quad t^* = \dfrac{0.772352 - 0.7712}{0.77408 - 0.7712} = 0.4$

$\therefore x_4 = 1 \quad l^{(4)} = 0.7712 \quad u^{(4)} = 0.773504$

Decodingsteps:

(1) Initialize

$l^{(0)} = 0$ and $u^{(0)} = 1$

(2) For each k

(2-1) find $t^* = \dfrac{tag - l^{(k-1)}}{u^{(k-1)} - l^{(k-1)}}$

(2-2) Find x_k s.t.

$F_X(x_k - 1) \le t^* < F_X(x_k)$

(2-3) update $u^{(k)}$ & $l^{(k)}$

Q:

1. 如何知道decode已結束?
2. 需用多少bits來表示tag?

Decodability and Efficiency of the Arithmetic Code (1)

- The binary representation of $\bar{T}_X(x)$ can be truncated to $l(x) = \left\lceil \log \dfrac{1}{P(x)} \right\rceil + 1$ bits

Symbol	P_X	F_X	$\bar{T}_X$	In binary	$\lceil \log(1/P(x)) \rceil + 1$	Code
1	1/3	1/3	1/6	0.00101…	3	001
2	1/6	1/2	5/12	0.01101…	4	0110
3	1/4	3/4	5/8	0.101	3	101
4	1/4	1	7/8	0.111	3	111

A binary code for a four-letter alphabet

Proof :

1. this code is unique (i.e. non - ambiguous code)

i.e. To show that $\lfloor \bar{T}_X(x) \rfloor_{l(x)} \in [F_X(x-1), F_X(x))$

(i) $\lfloor \bar{T}_X(x) \rfloor_{l(x)} < F_X(x)$ trival

Q: i.e. $\lfloor \bar{T}_X(x) \rfloor_{l(x)-1}$ 可能 $\notin [F_X(x-1), F_X(x))$

(ii) $\because 0 \le \bar{T}_X(x) - \lfloor \bar{T}_X(x) \rfloor_{l(x)} < \dfrac{1}{2^{l(x)}} \quad -(A)$

Decodability and Efficiency of the Arithmetic Code (2)

$$\text{and} \quad \frac{1}{2^{l(x)}} = \frac{1}{2^{\left\lceil \log \frac{1}{P(x)} \right\rceil + 1}} \le \frac{1}{2^{\log \frac{1}{P(x)} + 1}} = \frac{1}{2 \frac{1}{P(x)}} = \frac{P(x)}{2}$$

$$\because \quad \frac{P(x)}{2} = \bar{T}_X(x) - F_X(x-1)$$ Q: 如果 tag不是這樣取?

$$\therefore \quad \bar{T}_X(x) - F_X(x-1) \quad \ge \quad 2^{-\frac{1}{l(x)}} \quad \text{- (B)}$$

$$\therefore \quad \text{By (A) \& (B)}, \quad \left\lfloor \bar{T}_X(x) \right\rfloor_{l(x)} > F_X(x-1)$$

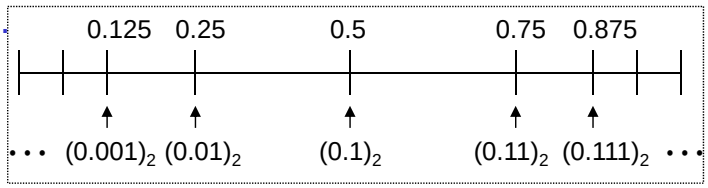
$$\therefore \quad \left\lfloor \bar{T}_X(x) \right\rfloor_{l(x)} \in \left[F_X(x-1), F_X(x) \right)$$

Decodability and Efficiency of the Arithmetic Code (3)

2. Uniquely decodable

Lemma: Given a number *a* in the interval [0,1) with an *n*-bit binary representation $[a_1 a_2 \ldots a_n]$, for any other number *b* to have a binary representation with $[a_1 a_2 \ldots a_n]$ as the prefix, *b* have to lie in the interval $[a, a+1/2^n)$.

a is prefix of b



To show that $\left[\left\lfloor \bar{T}_X(x) \right\rfloor_{l(x)}, \left\lfloor \bar{T}_X(x) \right\rfloor_{l(x)} + \frac{1}{2^{l(x)}} \right) \subset [F_X(x-1), F_X(x))$

(i) $\left\lfloor \bar{T}_X(x) \right\rfloor_{l(x)} > F_X(x-1)$ 已証

(ii) $F_X(x) - \left\lfloor \bar{T}_X(x) \right\rfloor_{l(x)} > F_X(x) - \bar{T}_X(x) = \frac{P(x)}{2} > \frac{1}{2^{l(x)}}$

i.e. this code is a prefix code

Suppose $\left\lfloor \bar{T}_X(x_1) \right\rfloor_{l(x_1)}$ is prefix of $\left\lfloor \bar{T}_X(x_2) \right\rfloor_{l(x2)}$

then $\left\lfloor \bar{T}_X(x_1) \right\rfloor_{l(x_1)} \le \left\lfloor \bar{T}_X(x_2) \right\rfloor_{l(x2)} < \left\lfloor \bar{T}_X(x_1) \right\rfloor_{l(x_1)} + \frac{1}{2^{l(x_1)}}$

But, $\left[\left\lfloor \bar{T}_X(x_1) \right\rfloor_{l(x_1)}, \left\lfloor \bar{T}_X(x_1) \right\rfloor_{l(x_1)} + \frac{1}{2^{l(x_1)}} \right) \subset [F_X(x_1-1), F_X(x_1))$

$\therefore \left\lfloor \bar{T}_X(x_2) \right\rfloor_{l(x2)} \in [F_X(x_1-1), F_X(x_1))$ $(\rightarrow\leftarrow)$

Decodability and Efficiency of the Arithmetic Code (3)

3. How efficient is it ?

$l_{A^m} = \sum P(x)l(x) = \sum P(x)\left[\lceil \log(1/P(x)) \rceil + 1\right]$,

where A^m : the number of sequences of length m

$< \sum P(x)\left[\log(1/P(x)) + 1 + 1\right] = -\sum P(x)\log P(x) + 2\sum P(x)$

$= H(X^{(m)}) + 2$, where $X^{(m)} \equiv \{x_1 x_2 \ldots\ldots\ldots x_m\}$

$\therefore\ H(X^{(m)}) \le l_{A^m} < H(X^{(m)}) + 2 \Rightarrow \dfrac{H(X^{(m)})}{m} \le l_A < \dfrac{H(X^{(m)})}{m} + \dfrac{2}{m}$

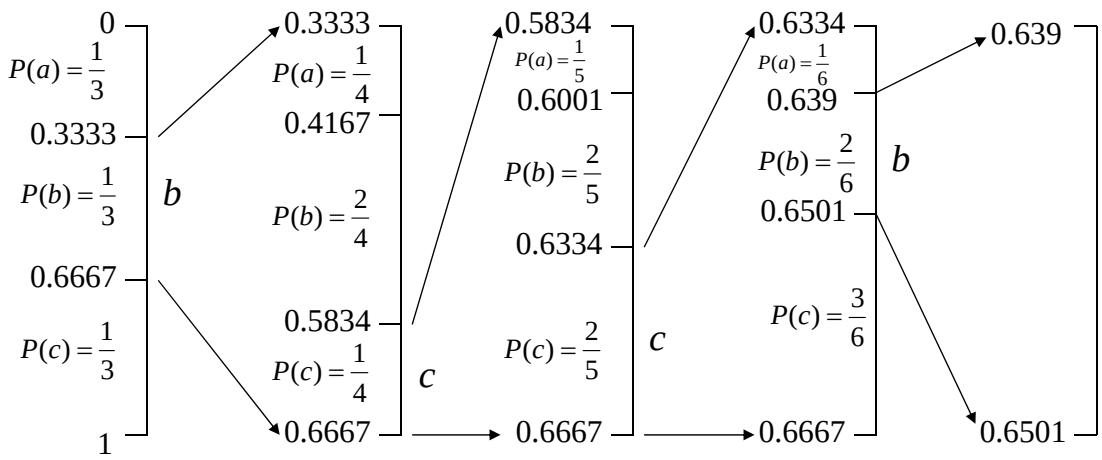
$\because$ iid asumption $H(X^{(m)}) = mH(X)$ $\therefore\ H(X) \le l_A < H(X) + \dfrac{2}{m}$

Compared with Huffman code $H(x) \le l_H < H(x) + \dfrac{1}{m}$

- 當 m 小時 , arithmetic code 比 Huffman code 差 e.g. 4.4.1 & 4.5.1
- m↑ 對 arithmetic code 較恰當
- easy to implement a system with multiple arithmetic codes
- much easier to adapt arithmetic codes to changing input statistics

Adaptive Arithmetic Coding (1)

- Example :
  - for a source alphabet {a,b,c}
  - input sequence = bccb



Algorithm Implementation

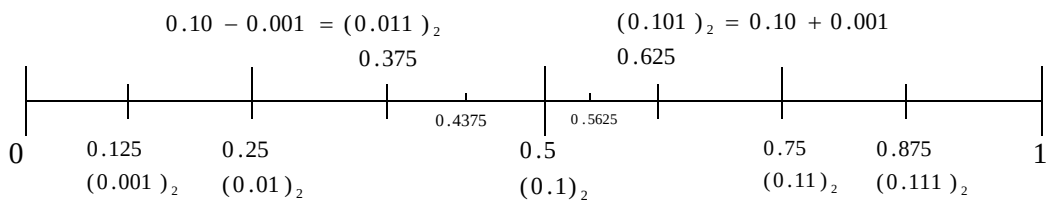
- Rescaling the interval
  ∵ the precision required to represent the interval grows with the length of the sequence
  ⇒ once the codec knows which portion contains the tag
  1. ignore the portion which not containing the tag
  2. concentrate on the portion containing the tag
  ⇒ map the interval containing the tag to [0,1)

- Incremental encoding & decoding
  ⇒ to generate portions of the code of the tag without waiting to see the entire sequence

- Integer arithmetic
  - to prevent loss of precision due to round-off error
  - to speed up arithmetic coding

Rescaling & Incremental Coding (1)

- Incremental encoding
  - send the *MSB* when $l^{(k)}$ & $u^{(k)}$ have a common prefix
- Rescaling the interval
  - map the half interval containing the tag to [0,1).



$E_1 : [0, 0.5) \rightarrow [0, 1)$ $(.0xxx.....)_2$

$E_1(x) = 2x$

e.g.:

$0.2 = (.\underline{00}11...)_2$ E1 $[0, 0.25)$ E1

$0.4 = (.011...)_2$ E1 $[0, 0.5)$

$0.8 = (.11...)_2$ E1 $[0, 1)$

$E_2 : [0.5, 1) \rightarrow [0, 1)$ $(.1xxx.....)_2$

$E_2(x) = 2(x-0.5)$

e.g. :

$0.8 = (.\underline{11}0011...)_2$ E2 $[0.75, 1)$ E2

$0.6 = (.10011...)_2$ E2 $[0.5, 1)$ E2

$0.2 = (.0011...)_2$ $[0, 1)$

Q: $[0.75, 0.875)$ --- $(.110xxx)_2$

Rescaling & Incremental Coding (2)

E_3 : [0.25 , 0.75) → [0 , 1) $(.01xxx\ldots)_2$ ~

$E_3(x) = 2(x - 0.25)$ $(.10xxx\ldots)_2$

(i) [0 , 1)

↑ E_2

[0.5 , 1) .1~

↑ E_3

[0.5 , 0.75) .10 ~

↑ E_3

[0.5 , 0.625) .100~

↑ E_3

[0.5 , 0.5625) .1000~

```
|————————————[-)-|—|———|
                ↓ E3
              [   )
                ↓E3
              [     )
                ↓E3
              [           )
                     ↓E2
[                          )
```

$x = 0.562 = (.100011\sim)_2$

$E_3E_3E_3E_2$

send 1 0 0 0

Rescaling & Incremental Coding (3)

(ii) e.g.:

[0 , 1)

↑ E_1

[0 , 0.5) .0 ~

↑ E_3

[0.25 , 0.5) .01 ~

↑ E_3

[0.375 , 0.5) .011~

↑ E_3

[0.4375, 0.5) .0111~

$x = 0.4376 = (.011100\sim)_2$

$E_3E_3E_3E_1$

send 0 1 1 1

```
|————————[—)————————————|
              ↓ E3
          [     )
            ↓E3
        [       )
          ↓E3
      [           )
            ↓E1
[                           )
```

(iii)

[0.3568 , 0.54112)

↓ E_3

[0.2136 , 0.54228)

x = 0.4 or 0.5

E_3

經過一次E_3後不做 E_1 or E_2 ∴此時tag第一個bit 可能 0 or 1

Incremental Encoding(E1&E2)(1)

Example : input sequence = $(a_1\, a_3\, a_2\, a_1)$

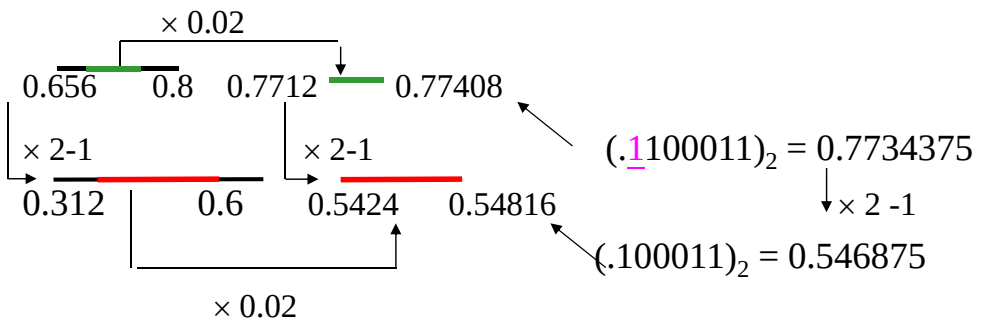
$F_X(0)=0$
$F_X(1)=0.8$
$F_X(2)=0.82$
$F_X(3)=1$

$$\begin{cases} l^{(0)} = 0 \\ u^{(0)} = 1 \end{cases} \xrightarrow{a_1} \begin{cases} l^{(1)} = 0 \\ u^{(1)} = 0.8 \end{cases} \xrightarrow{a_3} \begin{cases} l^{(2)} = 0.656 \\ u^{(2)} = 0.8 \end{cases}$$

send code = 1

間寬(距) $l^{(2)} = 2 \times (0.656 - 0.5) = 0.312$

加倍 $u^{(2)} = 2 \times (0.8 - 0.5) = 0.6$



Incremental Encoding(E1&E2)(2)

a_2

$$\begin{cases} l^{(3)} = 0.5424 \\ u^{(3)} = 0.54816 \end{cases}$$

原來之2倍寬

send code = 1

$$\begin{cases} l^{(3)} = 2 \times (0.5424 - 0.5) = 0.0848 \\ u^{(3)} = 2 \times (0.54816 - 0.5) = 0.09632 \end{cases}$$

send code = 0

$$\begin{cases} l^{(3)} = 2 \times 0.0848 = 0.1696 \\ u^{(3)} = 2 \times 0.09632 = 0.19264 \end{cases}$$

send code = 0

$$\begin{cases} l^{(3)} = 2 \times 0.1696 = 0.3392 \\ u^{(3)} = 2 \times 0.19264 = 0.38528 \end{cases}$$

send code = 0

$$\begin{cases} l^{(3)} = 2 \times 0.3392 = 0.6784 \\ u^{(3)} = 2 \times 0.38528 = 0.77056 \end{cases}$$

send code = 1

$$\begin{cases} l^{(3)} = 2 \times (0.6784 - 0.5) = 0.3568 \\ u^{(3)} = 2 \times (0.77056 - 0.5) = 0.54112 \end{cases}$$

a_1

$$\begin{cases} l^{(4)} = 0.3568 \\ u^{(4)} = 0.504256 \end{cases}$$

原來之64倍寬?

Use value 0.5=0.10…0

Send code = 10…0, add "0…0" stuffs to make 11000110…0 be words

Q : 最後Encode a_1 如果不送出 '1' 可以 送出? 01 X 0101(0.3125) X 011 (0.375) ˇ

Q : 如果input 只有$a_1a_3a_2$, encoder仍是送出? 1100011

∴最後 'a_1' symbol 不再需要bit encode

$(.10001)_2 = 0.53125$

$(.100011)_2 = 0.546875$

Incremental Decoding(E1&E2)(1)

Example : input bit stream = (11000110…0)

⇒ To begin the decoding process, we need to have enough information to decode the first symbol unambiguously, i.e. to guarantee unambiguously decoding, the number of bits received should point to an interval smaller than the smallest tag interval

$\begin{cases} l^{(0)} = 0 \\ u^{(0)} = 1 \end{cases}$ ⇒ t* = tag, decode a_1 ⇒ t* = $\dfrac{(0.765625 - 0)}{(0.8 - 0)}$ = 0.957

tag = $(.110001)_2$ =0.765625

$\begin{cases} l^{(1)} = 0 \\ u^{(1)} = 0.8 \end{cases}$

decode a_3

$\begin{cases} l^{(2)} = 0.656 \\ u^{(2)} = 0.8 \end{cases}$

$(.11)_2$=0. 75 ×
$(.101)_2$=0. 625 ×
$(.1001)_2$=0. 5625 ?
(but 0.8698% > 0.82 %)
$(.10001)_2$=0. 53125 ?

Encode a_2

$F_X(0)=0$
$F_X(1)=0.8$
$F_X(2)=0.82$
$F_X(3)=1$

$2^{-5} = 0.03125$
$2^{-6} = 0.015625$?
$2^{-k} < 0.02 \Rightarrow k = 6$

$\begin{cases} l^{(2)} = 2 \times (0.656 - 0.5) = 0.312 \\ u^{(2)} = 2 \times (0.8 - 0.5) = 0.6 \end{cases}$

tag = $(.100011)_2$ = 0.546875 (discard "1" bit)

Q: 如果僅取 5個 (2~4?) bit, a_1 及 a_3 會正確地被decode出, 而 a_2 則否.

Ans: $(.10001)_2$=0.53125

t* = $\dfrac{(0.53125 - 0.312)}{(0.6 - 0.312)}$ = 0.76128

∴decode a_1

Ans:

$(.1001)_2$=0.5625

t* = $\dfrac{(0.5625 - 0.312)}{(0.6 - 0.312)}$ = 0.86979

∴decode a_3

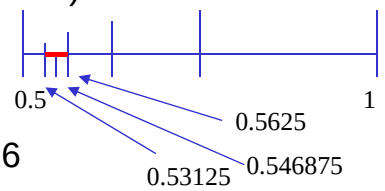
Incremental Decoding(E1&E2)(2)

$t^* = \dfrac{(0.546875-0.312)}{(0.6-0.312)}$

$= 0.8155$

decode a_2

$\begin{cases} l^{(3)} = 0.5424 \\ u^{(3)} = 0.54816 \end{cases}$



$\begin{cases} l^{(3)} = 2 \times (0.5424 - 0.5) = 0.0848 \\ u^{(3)} = 2 \times (0.54816 - 0.5) = 0.09632 \end{cases}$

tag = .000110 = 0.09375 (discard "1" bit)

$\begin{cases} l^{(3)} = 2 \times 0.0848 = 0.1696 \\ u^{(3)} = 2 \times 0.09632 = 0.19264 \end{cases}$

tag = .001100 = 0.1875 (discard "0" bit)

$\begin{cases} l^{(3)} = 2 \times 0.1696 = 0.3392 \\ u^{(3)} = 2 \times 0.19264 = 0.38528 \end{cases}$

tag = .011000 = 0.375 (discard "0" bit)

$\begin{cases} l^{(3)} = 2 \times 0.3392 = 0.6784 \\ u^{(3)} = 2 \times 0.38528 = 0.77056 \end{cases}$

tag = .110000 = 0.75 (discard "0" bit)

$\begin{cases} l^{(3)} = 2 \times (0.6784 - 0.5) = 0.3568 \\ u^{(3)} = 2 \times (0.77056 - 0.5) = 0.54112 \end{cases}$

tag = .100000 = 0.5 (discard "1" bit)

$t^* = \dfrac{(0.5-0.3568)}{0.54112-0.3568} = 0.7769$

decode a_1

$\begin{cases} l^{(4)} = 0.3568 \\ u^{(4)} = 0.504256 \end{cases}$

tag 位於[0.3568, 0.54112) 0~80%之間

(i.e. $0 \le t^* < 0.8$)

Incremental Encoding(E1,E2,&E3)

Example : input sequence = $(a_1\, a_3\, a_2\, a_1)$

$\begin{cases} l^{(0)} = 0 \\ u^{(0)} = 1 \end{cases}$ (counter = 0) $\xrightarrow{a_1}$ $\begin{cases} l^{(1)} = 0 \\ u^{(1)} = 0.8 \end{cases}$ $\xrightarrow{a_3}$ $\begin{cases} l^{(2)} = 0.656 \\ u^{(2)} = 0.8 \end{cases}$ $\xrightarrow{a_2}$

Send code = 1

$\begin{cases} l^{(2)} = 2 \times (0.656 - 0.5) = 0.312 \\ u^{(2)} = 2 \times (0.8 - 0.5) = 0.6 \end{cases}$

(counter = 1)

$\begin{cases} l^{(2)} = 2 \times (0.312 - 0.25) = 0.124 \\ u^{(2)} = 2 \times (0.6 - 0.25) = 0.7 \end{cases}$

Send code = 10 (counter = 0)

$\begin{cases} l^{(3)} = 2 \times (0.5848 - 0.5) = 0.1696 \\ u^{(3)} = 2 \times (0.59632 - 0.5) = 0.19264 \end{cases}$

Send code = 0

$\begin{cases} l^{(3)} = 2 \times 0.1696 = 0.3392 \\ u^{(3)} = 2 \times 0.19264 = 0.38528 \end{cases}$

Send code = 0

$\begin{cases} l^{(3)} = 2 \times 0.3392 = 0.6784 \\ u^{(3)} = 2 \times 0.38528 = 0.77056 \end{cases}$

Send code = 1

$\begin{cases} l^{(3)} = 2 \times (0.6784 - 0.5) = 0.3568 \\ u^{(3)} = 2 \times (0.77056 - 0.5) = 0.54112 \end{cases}$

(counter = 1)

$\begin{cases} l^{(3)} = 2 \times (0.3568 - 0.25) = 0.2136 \\ u^{(3)} = 2 \times (0.54112 - 0.25) = 0.58224 \end{cases}$

$\xrightarrow{a_1}$ $\begin{cases} l^{(4)} = 0.2136 \\ u^{(4)} = 0.508512 \end{cases}$

use value 0.5 = 0.10…0

send code = 10…0 or 011…0

$\begin{cases} l^{(3)} = 0.5848 \\ u^{(3)} = 0.59632 \end{cases}$ 原來之 4 倍寬 [0.7712, 0.77408)

$\xrightarrow{a_1}$ $\begin{cases} l^{(4)} = 0.3568 \\ u^{(4)} = 0.504256 \end{cases}$

經過一次E3後, 不作 E1 or E2

a_1a_3 : [0.656,0.8) :

$(0.1\underline{01})_2=0.625$ $(0.1\underline{01}1)_2=0.6875$

$(0.1\underline{10})_2=0.75$

$a_1a_3a_2$: [0.7712,0.77408) :

$(0.11000\underline{101})_2=0.76953125$ $(0.11000\underline{101}1)_2=0.771484375$

$(0.11000\underline{110})_2=0.7734375$

$a_1a_3a_2a_1$: [0.7712,0.773504) :

Incremental Decoding(E1,E2,&E3)

Example : input bit stream = (11000110…0)

$\begin{cases} l^{(0)} = 0 \\ u^{(0)} = 1 \end{cases}$

B = $(110001)_2$

tag =0.765625

t* = tag

decode a_1

$\begin{cases} l^{(1)} = 0 \\ u^{(1)} = 0.8 \end{cases}$

$t^* = \dfrac{(0.765625 - 0)}{(0.8 - 0)} = 0.957$

decode a_3

$\begin{cases} l^{(2)} = 0.656 \\ u^{(2)} = 0.8 \end{cases}$

$\begin{cases} l^{(2)} = 2 \times (0.656 - 0.5) = 0.312 \\ u^{(2)} = 2 \times (0.8 - 0.5) = 0.6 \end{cases}$

B = $(100011)_2$ tag = 0.546875 (discard "1" bit)

(counter = 1)

$\begin{cases} l^{(2)} = 2 \times (0.312 - 0.25) = 0.124 \\ u^{(2)} = 2 \times (0.6 - 0.25) = 0.7 \end{cases}$

tag = (tag-0.25) ×2 =$(.100110)_2$ = 0.59375

not

$t^* = \dfrac{(0.59375 - 0.124)}{(0.7 - 0.124)} = 0.8155$

decode a_2

$\begin{cases} l^{(3)} = 0.5848 \\ u^{(3)} = 0.59632 \end{cases}$

(discard "10" bits)

B = $(001100)_2$

(counter = 0)

$\begin{cases} l^{(3)} = 2 \times (0.5848-0.5) = 0.1696 \\ u^{(3)} = 2 \times (0.59632-0.5) = 0.19264 \end{cases}$

tag = .001100 = 0.1875 (discard "0" bit)

B = $(011000)_2$

$\begin{cases} l^{(3)} = 2 \times 0.1696 = 0.3392 \\ u^{(3)} = 2 \times 0.19264 = 0.38528 \end{cases}$

tag = .011000 = 0.375 (discard "0" bit)

B = $(110000)_2$

$\begin{cases} l^{(3)} = 2 \times 0.3392 = 0.6784 \\ u^{(3)} = 2 \times 0.38528 = 0.77056 \end{cases}$

tag = .110000 = 0.75 (discard "1" bit)

B = $(100000)_2$

$\begin{cases} l^{(3)} = 2 \times (0.6784-0.5) = 0.3568 \\ u^{(3)} = 2 \times (0.77056-0.5) = 0.54112 \end{cases}$

tag = .100000 = 0.5

(counter = 1)

$\begin{cases} l^{(3)} = 2 \times (0.3568 - 0.25) = 0.2136 \\ u^{(3)} = 2 \times (0.54112 - 0.25) = 0.58224 \end{cases}$

tag = (tag-0.25) ×2 =$(.100000)_2$ = 0.5

$t^* = \dfrac{(0.5-0.2136)}{0.58224-0.2136} = 0.7769$

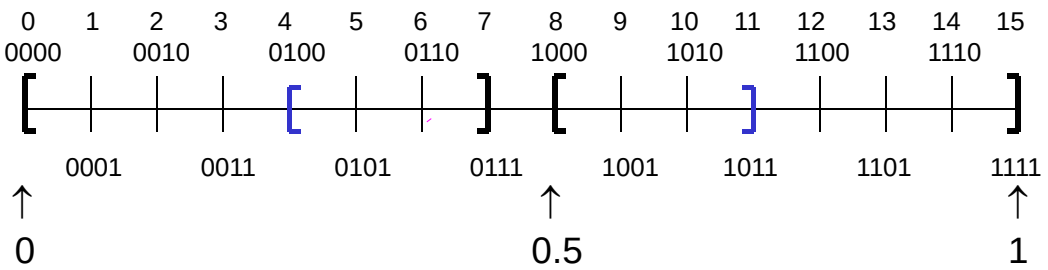
decode a_1

$\begin{cases} l^{(4)} = 0.2136 \\ u^{(4)} = 0.508512 \end{cases}$

Integer Implementation (1)

12/16=0.75
1/16=0.0625
3/16=0.1875

13/16=0.8125
1/16=0.0625
2/16=0.125



$E_1 : [0 , 7] \rightarrow [0 , 15]$ $E_2 : [8 , 15] \rightarrow [0 , 15]$ $E_3 : [4 , 11] \rightarrow [0 , 15]$

$E_1 : [0, 2^{n-1}-1] \rightarrow [0, 2^n-1]$ $E_2 : [2^{n-1}, 2^n-1] \rightarrow [0 , 2^n-1]$ $E_3 : [2^{n-2}, 3\times 2^{n-2}-1] \rightarrow [0 , 2^n-1]$

$$\begin{cases} l^{(n)} = l^{(n-1)} + \lfloor (u^{(n-1)} - l^{(n-1)} + 1)\, F_x(x_n - 1) \rfloor \\ u^{(n)} = l^{(n-1)} + \lfloor (u^{(n-1)} - l^{(n-1)} + 1)\, F_x(x_n) \rfloor - 1 \end{cases}$$

E1 : $[x,y] \rightarrow [2x,2y+1]$

E2 : $[x,y] \rightarrow [2(x-2^{n-1}), 2(y-2^{n-1})+1]$

E3 : $[x,y] \rightarrow [2(x-2^{n-2}), 2(y-2^{n-2})+1]$

Q1 : 是否可能 $u^{(n)} < l^{(n)}$?(bit數取太少時)

EX.

$$X_1=1 \begin{cases} l^{(1)} = 0 + \lfloor (15 - 0 + 1) \times 0 \rfloor = 0 \\ u^{(1)} = 0 + \lfloor (15 - 0 + 1) \times 0.8 \rfloor - 1 = 11 \end{cases}$$

$$X_1=3 \begin{cases} l^{(1)} = 0 + \lfloor (15 - 0 + 1) \times 0.82 \rfloor = \lfloor 13.12 \rfloor = 13 \\ u^{(1)} = 0 + \lfloor (15 - 0 + 1) \times 1 \rfloor - 1 = 15 \end{cases}$$

$a_1 : [0 ,11]$ $P(a_1)=0.8$
$a_2 : [12 ,12]$ $P(a_2)=0.02$
$a_3 : [13 ,15]$ $P(a_3)=0.18$

Q2: compression loss

Q1: If $l^{(0)} = 7$, $u^{(0)}=12$; or $l^{(0)} = 3$, $u^{(0)}=8$

$$X_1=1 \begin{cases} l^{(1)} = 7 + \lfloor (12 - 7 + 1) \times 0 \rfloor = 7 \\ u^{(1)} = 7 + \lfloor (12 - 7 + 1) \times 0.8 \rfloor - 1 = 10 \end{cases}$$

$$X_1=2 \begin{cases} l^{(1)} = 7 + \lfloor (12 - 7 + 1) \times 0.8 \rfloor = 11 \\ u^{(1)} = 7 + \lfloor (12 - 7 + 1) \times 0.82 \rfloor - 1 = 10 \end{cases}$$

$$X_1=3 \begin{cases} l^{(1)} = 7 + \lfloor (12 - 7 + 1) \times 0.82 \rfloor = 11 \\ u^{(1)} = 7 + \lfloor (12 - 7 + 1) \times 1 \rfloor - 1 = 12 \end{cases}$$

Q2:

$$E = -0.8\log 0.8 - 0.02\log 0.02 - 0.18\log 0.18$$

$$E_1 = -0.8\log 0.75 - 0.02\log 0.0625 - 0.18\log 0.1875 \qquad E_1 > E\ ?$$

$$E_2 = -0.8\log 0.8125 - 0.02\log 0.0625 - 0.18\log 0.125 \qquad E_2 > E\ ?$$

Integer Implementation (2)

0 16 31 32 35 39 47 63

(1) E_1 : [0 , 31] → [0 , 63]

e.g. [001010 , 011111]
10 31
[010100 , 111111]
20 63

點數增為兩倍

E_1, E_2 :

(*) shift *l* to the left by 1 bit & shift 0 into LSB
shift *u* to the left by 1 bit & shift 1 into LSB

(2) E_2 : [32 , 63] → [0 , 63]

e.g. [100001 , 110110]
33 54
[000010 , 101101]
2 45

E_3 :

(1) complement the 2nd MSB of *l* & *u*

(2) (*)

(3) E_3 : [16 , 47] → [0 , 63]

eg. [100000 , 100010] → [100000 , 100101] →
32 34 E_3 32 37 E_3
[100000 , 101011] → [100000 , 110111] → send 1000
32 43 E_3 32 55 E_2

Incremental Encoding (Integer) (1)

Example : input sequence = $(a_1\, a_3\, a_2\, a_1)$ E_1: [0 , 127] E_2 : [128 , 255] E_3 : [64 , 191]

$l^{(0)} = 0 = (00000000)_2$
$u^{(0)} = 255 = (11111111)_2$

$\xrightarrow{a_1}$

$l^{(1)} = 0 = (00000000)_2$
$u^{(1)} = 203 = (11001011)_2$

$\xrightarrow{a_3}$

$l^{(2)} = 167 = (10100111)_2$
$u^{(2)} = 203 = (11001011)_2$
send code = 1

E2

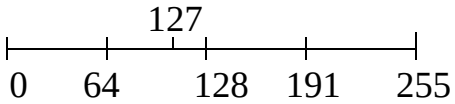
$l^{(2)} = (01001110)_2 = 78$
$u^{(2)} = (10010111)_2 = 151$
scale3 = 1

E3

$l^{(2)} = (00011100)_2 = 28$
$u^{(2)} = (10101111)_2 = 175$

count(a_1) = 40
count(a_2) = 1
count(a_3) = 9

80/100, 2/100, 18/100
40/50, 1/50, 9/50



0 64 127 128 191 255

Q : why m = 8 , $(1/4 \times 2^m) + 2 \geq 50$

因當最小可能區間為[127 , 192] 或 [63 , 128]時, 仍不需 作 rescaling

Incremental Encoding (Integer) (2)

a_2

E2 $\begin{cases} l^{(3)} = 146 = (10010010)_2 \\ u^{(3)} = 148 = (10010100)_2 \end{cases}$
send code = 10 scale3 = 0

E1 $\begin{cases} l^{(3)} = (00100100)_2 = 36 \\ u^{(3)} = (00101001)_2 = 41 \end{cases}$
send code = 0

E1 $\begin{cases} l^{(3)} = (01001000)_2 = 72 \\ u^{(3)} = (01010011)_2 = 83 \end{cases}$
send code = 0

E2 $\begin{cases} l^{(3)} = (10010000)_2 = 144 \\ u^{(3)} = (10100111)_2 = 167 \end{cases}$
send code = 1

E1 $\begin{cases} l^{(3)} = (00100000)_2 = 32 \\ u^{(3)} = (01001111)_2 = 79 \end{cases}$
send code = 0

E3 $\begin{cases} l^{(3)} = (01000000)_2 = 64 \\ u^{(3)} = (10011111)_2 = 159 \end{cases}$
scale3 = 1

$\begin{cases} l^{(3)} = (00000000)_2 = 0 \\ u^{(3)} = (10111111)_2 = 191 \end{cases}$

a_1

$\begin{cases} l^{(4)} = (00000000)_2 = 0 \\ u^{(4)} = (10011000)\ 2 = 152 \end{cases}$

use value $l^{(4)} = 0$
send code = 1100010010000000

Q: 如果送出 '100000000' ?

Incremental Decoding (Integer)

- Example : input sequence = 1100010010000000

B= $(11000100)_2$ t = 196

E_1: [0 , 127] E_2 : [128 , 255] E_3 : [64 , 191]

$\begin{cases} l^{(0)} = 0 = (00000000)_2 \\ u^{(0)} = 255 = (11111111)_2 \end{cases}$ → $\begin{cases} l^{(1)} = 0 = (00000000)_2 \\ u^{(1)} = 203 = (11001011)_2 \end{cases}$ → t* = 48
decode a_3

$$t^* = \left\lfloor \frac{(t - l^{(0)} + 1) \times \text{TotalCount} - 1}{u^{(0)} - l^{(0)} + 1} \right\rfloor$$

$$= \left\lfloor \frac{197 \times 50 - 1}{255 - 0 + 1} \right\rfloor = 38$$

decode a_1 (since 0≤38<40)

Cum-count = $\begin{bmatrix} 0 \\ 40 \\ 41 \\ 50 \end{bmatrix}$

a_1: [0,203]
a_2: [204, 208]
a_3: [209,255]

E2 $\begin{cases} l^{(2)} = 167 = (10100111)_2 \\ u^{(2)} = 203 = (11001011)_2 \end{cases}$
(discard "1" bit)

E3 scale3=1 $\begin{cases} l^{(2)} = (01001110)_2 = 78 \\ u^{(2)} = (10010111)_2 = 151 \end{cases}$
B = $(10001001)_2$ t =137

$\begin{cases} l^{(2)} = (00011100)_2 = 28 \\ u^{(2)} = (10101111)_2 = 175 \end{cases}$
t = $(10010010)_2$ = 146

not

t* = 40 ⇒ decode a_2

E2 (discard "10" bit) scale3=0 $\begin{cases} l^{(3)} = 146 = (10010010)_2 \\ u^{(3)} = 148 = (10010100)_2 \end{cases}$

E1

$l^{(3)} = 36 = (00100100)_2$
$u^{(3)} = 41 = (00101001)_2$

$B = (00100100)_2$ $t = 36$

(discard "0" bit)

E1

$l^{(3)} = 72 = (01001000)_2$
$u^{(3)} = 83 = (01010011)_2$

$t = (01001000)_2 = 72$

(discard "0" bit)

E2

$l^{(3)} = 144 = (10010000)_2$
$u^{(3)} = 167 = (10100111)_2$

$t = (10010000)_2 = 144$

(discard "1" bit)

E1

$l^{(3)} = 32 = (00100000)_2$
$u^{(3)} = 79 = (01001111)_2$

$t = (00100000)_2 = 32$

(discard "0" bit)

E3

$l^{(3)} = 64 = (01000000)_2$
$u^{(3)} = 159 = (10011111)_2$

$t = (01000000)_2 = 64$

scale3=1

$l^{(3)} = 0 = (00000000)_2$
$u^{(3)} = 191 = (10111111)_2$

$t = (00000000)_2 = 0$

$l^{(3)} = 0 = (00000000)_2$
$u^{(3)} = 152 = (10011000)_2$

decode a_1

$t^* = 0$