

Lecture 7: Arithmetic Coding

Prof. Ja-Ling Wu

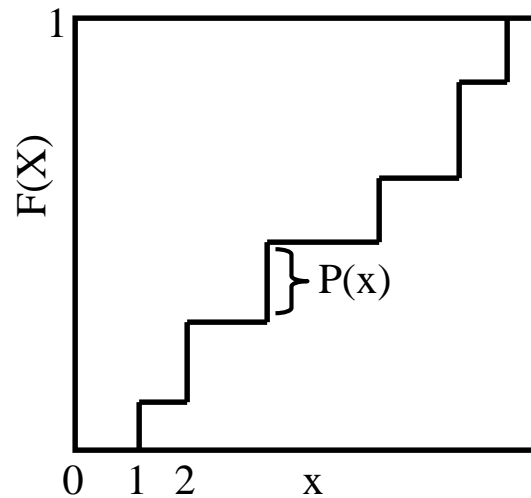
Department of Computer Science
and Information Engineering
National Taiwan University



Shannon-Fano-Elias Coding

- W.l.o.g. we can take $\mathbf{X}=\{1,2,\dots,m\}$. Assume $p(x)>0$ for all x . The **cumulative distribution function** $F(x)$ is defined as

$$F(x) = \sum_{a \leq x} p(a)$$



- Consider the modified cumulative distribution function

$$\bar{F}(x) = \sum_{a < x} P(a) + \frac{1}{2} P(x)$$

where $\bar{F}(x)$ denotes the sum of the probabilities of all symbols less than x plus half the probability of the symbol x . Since the r.v. is discrete, the cumulative distribution function consists of steps of size $p(x)$. The value of the Function $\bar{F}(x)$ is the midpoint of the step corresponding to x .

Since all the probabilities are positive, $F(a) \neq F(b)$ if $a \neq b$, and hence we can determine x if we know $\bar{F}(x)$. Thus the value of $\bar{F}(x)$ can be used as a code for x .



But in general $\bar{F}(x)$ is a **real number** expressible only by an **infinite number of bits**. So it is not efficient to use the exact value of $\bar{F}(x)$ as a code for x . If we use an approximate value, what is the required accuracy?

Assume that we **round off** $\bar{F}(x)$ to **$l(x)$ bits** (denoted by $\lfloor \bar{F}(x) \rfloor_{l(x)}$). Thus we use the first $l(x)$ bits of $\bar{F}(x)$ as a code for x .

By definition of **rounding off**, we have

$$\bar{F}(x) - \lfloor \bar{F}(x) \rfloor_{l(x)} < \frac{1}{2^{l(x)}}$$

if $l(x) = \lceil \log \frac{1}{p(x)} \rceil + 1$, then

$$\frac{1}{2^{l(x)}} < \frac{p(x)}{2} = \bar{F}(x) - F(x-1)$$

and therefore $\lfloor \bar{F}(x) \rfloor_{l(x)}$ lies within the step corresponding to x . Thus **$l(x)$ bits suffice to describe x .**



- In addition to requiring that the codeword identify to corresponding symbol, we also require the set of codewords to be **prefix-free**.

Consider each codeword z_1, z_2, \dots, z_l to represent not a point but the interval $[0. z_1 z_2 \dots z_l, 0. z_1 z_2 \dots z_l + 1/2^l]$. The code is prefix-free iff the **intervals** corresponding to codewords are **disjoint**.

The **interval** corresponding to any codeword has length $2^{-l(x)}$, which is less than half the height of the step corresponding to x . The lower end of the interval is in the lower half of the step. Thus the upper end of the interval lies below the top of the step, and the interval corresponding to any codeword lies entirely within the step corresponding to that symbol in the cumulative distribution function. Therefore, the **intervals** corresponding to **different codewords** are **disjoint** and the code is prefix-free.

Note that this procedure does not require the symbols to be ordered in terms of probability.



- Since we use $l(x) = \lceil \log \frac{1}{p(x)} \rceil + 1$ bits to represent x , the expected length of this code is

$$L = \sum_x p(x)l(x) = \sum_x p(x) \left(\lceil \log \frac{1}{p(x)} \rceil + 1 \right) < H(x) + 2$$

if the probabilities are **ordered** as $p_1 \geq p_2 \geq \dots \geq p_m$ then

$$H(x) \leq L \leq H(x) + 1$$

Example:

x	$p(x)$	$F(x)$	$\bar{F}(x)$	$\bar{F}(x)$ in binary	$l(x) = \lceil \log \frac{1}{p(x)} \rceil + 1$	codeword
1	0.25	0.25	0.125	0.001	3	001
2	0.25	0.5	0.375	0.011	3	011
3	0.2	0.7	0.6	0.10011	4	1001
4	0.15	0.85	0.775	0.1100011	4	1100
5	0.15	1.0	0.925	0.1110110	4	1110



Arithmetic coding

- **Huffman coding** : a **bottom-up procedure**
the calculation of the probabilities of all source sequences of a particular block length and the construction of the corresponding **complete code tree**.

A better scheme is one which can be easily extended to longer block lengths **without having to redo all the calculations**.

Arithmetic coding : a direct extension of **Shannon-Fano-Elias coding** calculate the probability mass function $p(x^n)$ and the **cumulative distribution function** $F(x^n)$ for the source sequence x^n .

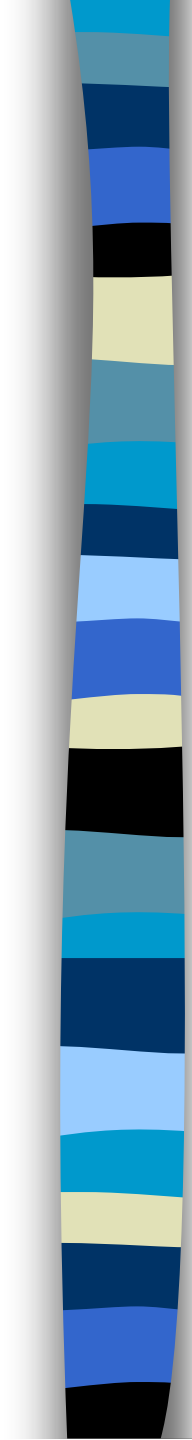


- We can use a number in the interval $(F(x^n) - P(x^n), F(x^n)]$ as the code for x^n . ← Shannon-Fano-Elias code
- For example:
expressing $F(x^n)$ to an accuracy of $\lceil \log \frac{1}{P(x^n)} \rceil$ will give us a code for the source.
 - ⇒ the codeword corresponding to any sequence lies within the step in the cumulative distribution function.
 - ⇒ The codewords are different sequences of length n . However, this procedure does not guarantee that the set of codewords is **prefix-free**.



- We can construct a **prefix-free** set by using $\overline{F}(x)$ **rounded off** to $\lceil \log \frac{1}{P(x)} \rceil + 1$ bits, as in the previous example.
- Arithmetic coding : Keep track of $F(x^n)$ and $P(x^n)$
We assume that we have a **fixed block length n** that is known to both the encoder and the decoder. With a small loss of generality, we assume that the source alphabet is binary. We assume that we have a simple procedure to calculate $P(x_1, x_2, \dots, x_n)$ for any string x_1, x_2, \dots, x_n .
We use the **natural lexicographic order** on strings, so a string x is **greater than** a string y if $x_i=1$ and $y_i=0$ for the first i such that $x_i \neq y_i$. Equivalently,



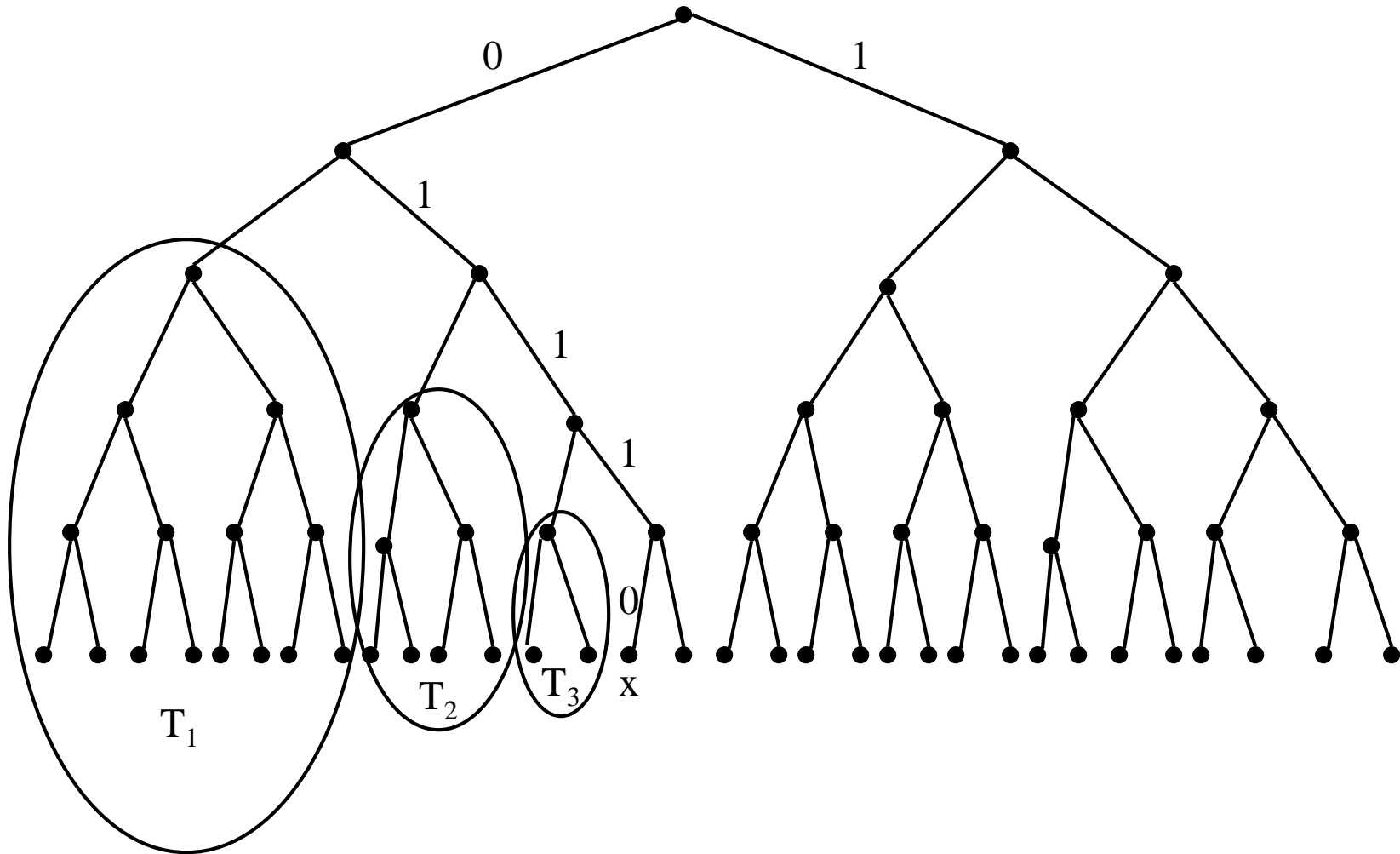

$$x > y \quad \text{if} \quad \sum_i x_i 2^{-i} > \sum_i y_i 2^{-i},$$

i.e., if the corresponding binary numbers satisfy

$$0 \cdot x > 0 \cdot y$$

We can arrange **the string as the leaves of a tree of depth n** , where each level of the tree corresponds to one bit.





- In the above tree, the ordering $x > y$ corresponds to the fact that x is to the right of y on the same level of the tree.



- The sum of the probabilities of all the leaves to the left of x^n is the sum of the probabilities of all the subtrees to the left of x^n .
- Let $T_{x_1 x_2 \dots x_{k-1} 0}$ be a subtree starting with $x_1 x_2 \dots x_{k-1} 0$. The probability of this subtree is

$$\begin{aligned}
 P(T_{x_1 x_2 \dots x_{k-1} 0}) &= \sum_{y_{k+1} \dots y_n} P(x_1 x_2 \dots x_{k-1} 0 y_{k+1} \dots y_n) \\
 &= P(x_1 x_2 \dots x_{k-1} 0)
 \end{aligned}$$

Therefore we can rewrite $F(x^n)$ as

$$\begin{aligned}
 F(x^n) &= \sum_{y^n \leq x^n} P(y^n) = \sum_{T: T \text{ is to the left of } x^n} P(T) \\
 &= \sum P(x_1 x_2 \dots x_{n-1} 0)
 \end{aligned}$$

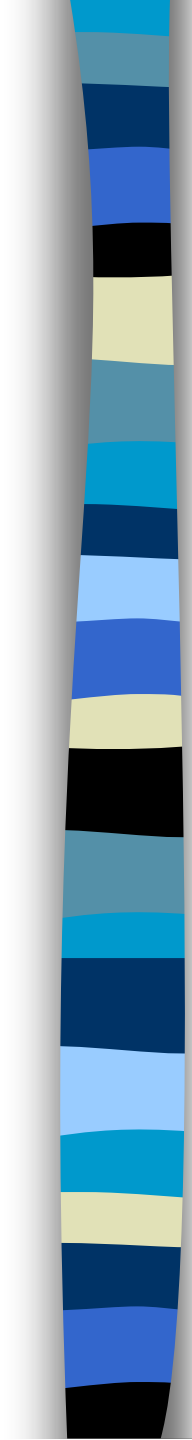


- Ex: if $P(0)=1-\theta$, $P(1)=\theta$ in the above Binary tree, then

$$\begin{aligned} F(01110) &= P(T_1) + P(T_2) + P(T_3) \\ &= P(00) + P(010) + P(0110) \\ &= (1-\theta)^2 + \theta(1-\theta)^2 + \theta^2(1-\theta)^2. \end{aligned}$$

To encode the **next bit** of the source sequence, we need only **calculate** $P(x^i x_{i+1})$ and **update** $F(x^i x_{i+1})$ using the above scheme. Encoding can therefore be done sequentially, by looking at the bits as they come in.



- 
- To **decode** the sequence, we **use the same procedure** to calculate the cumulative distribution function and check whether it exceeds the value corresponding to the codeword. We then use the above binary tree as a **decision tree**. At the top node, we check to see if the received codeword $F(x^n)$ is greater than $P(0)$. If it is, then the subtree starting with 0 is to the left of x^n and hence $x_1=1$. Continuing this process down the tree, we can decode the bits in sequence.

⇒ Thus we can compress and decompress a source sequence in a **sequential** manner.



- The above procedure depends on a model for which we can easily compute $P(x^n)$. Two examples of such models are **i.i.d. source**, where

$$P(x^n) = \prod_{i=1}^n P(x_i)$$

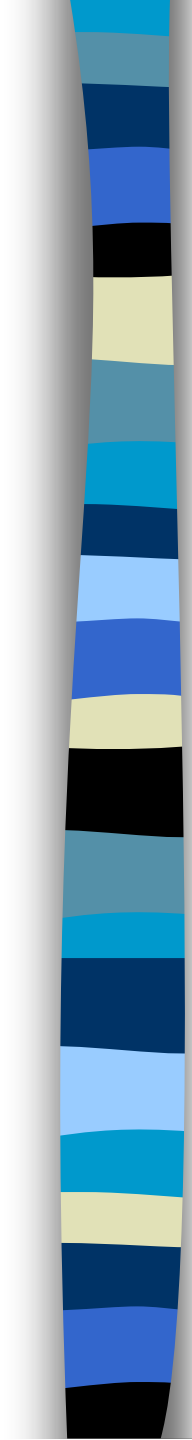
and **Markov source**, where

$$P(x^n) = P(x_1) \prod_{i=2}^n P(x_i | x_{i-1})$$

In both cases, we can easily compute $P(x^n | x_{n+1})$ from $P(x^n)$.

- Note that it is not essential that the probabilities used in the encoding be equal to the true distribution of the source.



- 
- In some cases, such as in image compression, it is difficult to describe a “true” distribution for the source. Even then, it is possible to apply the above arithmetic coding procedure.
 - G.G. Langdon. “An introduction to arithmetic coding,” IBM Journal of Research and Development, vol. 28, pp. 135-149, 1984.



Competitive Optimality of the Shannon code

- Theorem: Let $l(x)$ be the codeword lengths associated with the Shannon code and let $l'(x)$ be the codeword lengths associated with any other code. Then

$$P_r(l(x) \geq l'(x) + c) \leq \frac{1}{2^{c-1}}$$

Proof:

$$\begin{aligned} P_r(l(x) \geq l'(x) + c) &= P_r\left(\left\lceil \log \frac{1}{p(x)} \right\rceil \geq l'(x) + c\right) \\ &\leq P_r\left(\log \frac{1}{p(x)} \geq l'(x) + c - 1\right) \\ &= P_r\left(p(x) \leq 2^{-l'(x) - c + 1}\right) \\ &= \sum_{x: p(x) \leq 2^{-l'(x) - c + 1}} p(x) \\ &\leq \sum_x 2^{-l'(x)} 2^{-(c-1)} \\ &\leq 2^{-(c-1)}, \end{aligned}$$

Since $\sum 2^{-l'(x)} \leq 1$ by the **Kraft inequality**.

⇒ **No other code can do much better than the Shannon code in terms of the time.**





■ **Theorem:**

For a **dyadic probability mass function** $p(x)$, let $l(x) = \log \frac{1}{p(x)}$ be the word-lengths of the binary Shannon code for the source, and let $l'(x)$ be the length of any other **uniquely decodable binary code** for the source. Then

$$P_r(l(x) < l'(x)) \geq P_r(l(x) \geq l'(x))$$

with equality iff $l'(x) = l(x)$ for all x . \Rightarrow

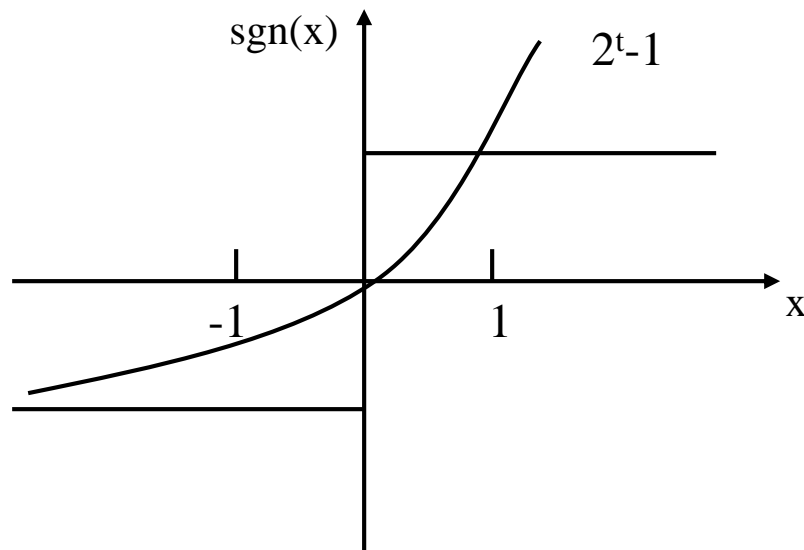
The code-length assignment $l(x) = \log \frac{1}{p(x)}$ is uniquely competitively optimal.



■ Proof:

Define the function $\text{sgn}(t)$ as follows:

$$\text{sgn}(t) = \begin{cases} 1, & \text{if } t > 0 \\ 0, & \text{if } t = 0 \\ -1, & \text{if } t < 0 \end{cases}$$



Note: $\text{sgn}(t) \leq 2^t - 1$ for $t = 0, \pm 1, \pm 2, \dots$

True for integer value of t



$$\begin{aligned}
& P_r(l'(x) < l(x)) - P_r(l'(x) > l(x)) \\
&= \sum_{x:l'(x) < l(x)} p(x) - \sum_{x:l'(x) > l(x)} p(x) \\
&= \sum_x p(x) \operatorname{sgn}(l(x) - l'(x)) \\
&= E \operatorname{sgn}(l(x) - l'(x)) \\
&\leq \sum_x p(x) (2^{l(x) - l'(x)} - 1) \\
&= \sum_x 2^{-l(x)} (2^{l(x) - l'(x)} - 1) \\
&= \sum_x 2^{-l'(x)} - \sum_x 2^{-l(x)} \\
&= \sum_x 2^{-l'(x)} - 1 \\
&\leq 1 - 1 = 0
\end{aligned}$$



- Corollary:
For **non-dyadic** probability mass function

$$E \operatorname{sgn}(l(x) - l'(x) - 1) \leq 0$$

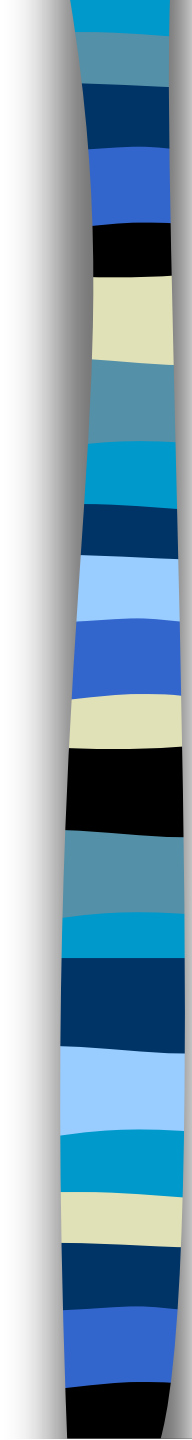
where $l(x) = \lceil \log \frac{1}{p(x)} \rceil$ and $l'(x)$ is any other code for the source.



Disadvantages of Huffman Codes

- The Huffman coder generates a new codeword for each input symbol.
 - the lower limit on compression for a Huffman coder is one bit per input symbol
- Higher compression ratio can be achieved by combining several symbols into a single unit ; however, the corresponding complexity for codeword construction will be increased
- Another problem with Huffman coding is that the coding and modeling steps are combined into a single process, and thus adaptive coding is difficult.
 - If the probabilities of occurrence of the input symbols change, then one has to redesign the Huffman table from scratch.





Arithmetic coding is a lossless compression technique that benefits from treating multiple symbols as a single data unit but at the same time retains the incremental symbol-by-symbol coding approach of Huffman coding. Arithmetic coding separates the coding from modeling. This allows for the dynamic adaptation of the probability model without affecting the design of the coder.

Encoding Process :

AR coding : a single codeword is assigned to each possible data set.

each codeword can be considered a **half-open subinterval** in the interval $[1, 0)$



By assigning **enough precision bits** to each of the codewords, one can distinguish one subinterval from any other subintervals, and thus uniquely decode the corresponding data set.

Like Huffman codewords, the more probable data sets correspond to larger subintervals and thus require fewer bits of precision.

Ex :

symbol	Probability	Huffman codeword
k	0.05	10101
l	0.2	01
u	0.1	100
w	0.05	10100
e	0.3	11
r	0.2	00
?	0.1	1011

input string :

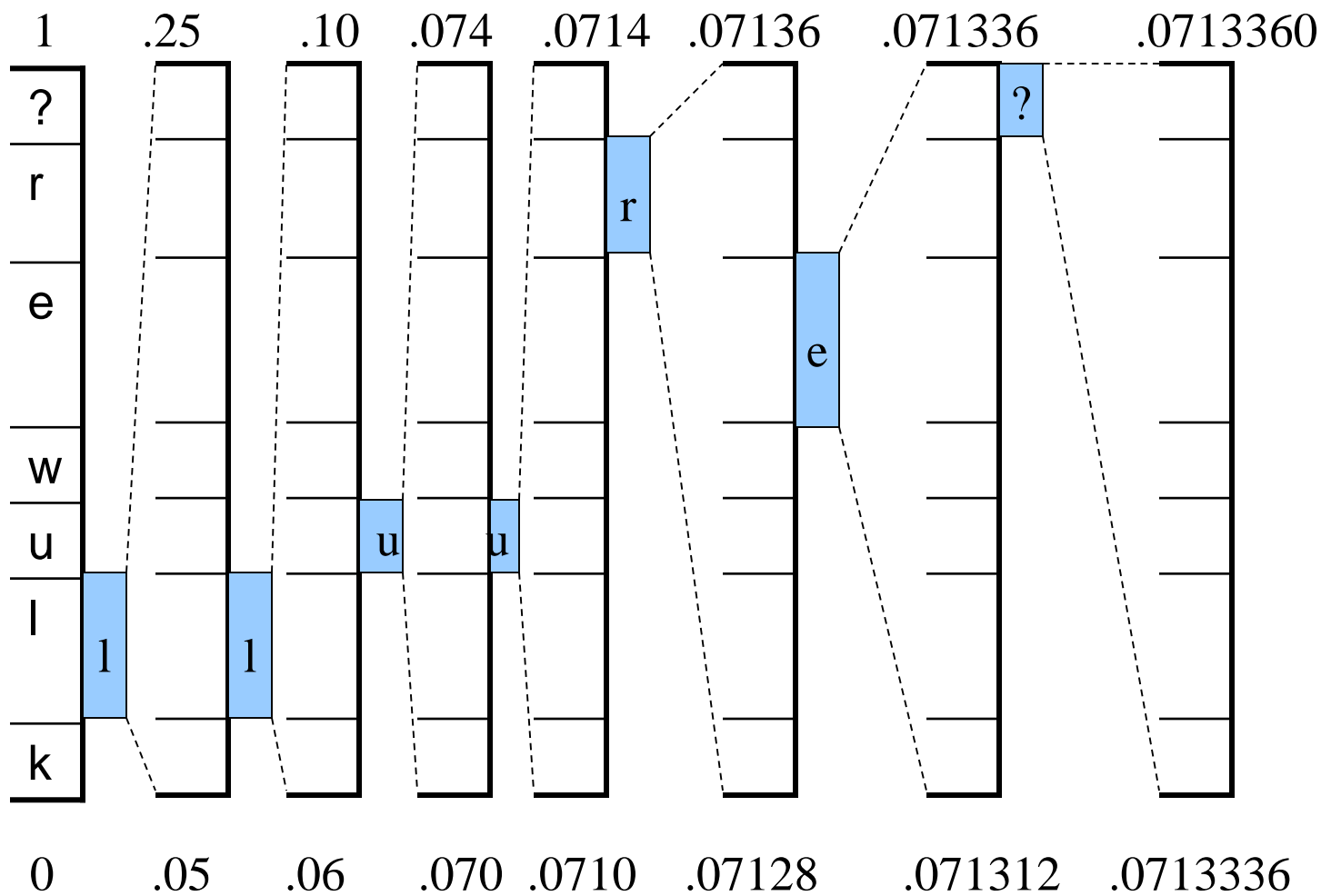
l l u u r e ?

01,01,100,100,00,11,1101

18 bits



Input



1. At the start of the process, the message is assumed to be in the half-open interval $[0,1)$. The interval is split into several subintervals, one subinterval for each symbol in our alphabet.

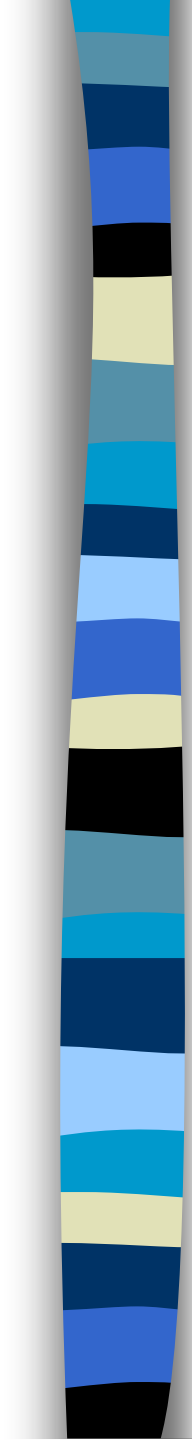
The upper limit of each subinterval is the cumulative probability up to and including the corresponding symbol.

The lower limit is the cumulative probability up to but not including the symbol.

S_i	P_i	subinterval
k	0.05	$[0.00, 0.05)$
l	0.2	$[0.05, 0.25)$
u	0.1	$[0.25, 0.35)$
w	0.05	$[0.35, 0.40)$
e	0.3	$[0.40, 0.70)$
r	0.2	$[0.70, 0.90)$
?	0.1	$[0.90, 1.00)$

end of message marker →





2. When the first symbol, I appears, we select the corresponding subinterval and make it the new current interval. The intervals of the remaining symbols in the alphabet set are then scaled accordingly.

Let $\text{Previous}_{\text{low}}$ and $\text{Previous}_{\text{high}}$ be the lower and the upper limit for the old interval.

Let $\text{Range} = \text{Previous}_{\text{low}} - \text{Previous}_{\text{high}}$

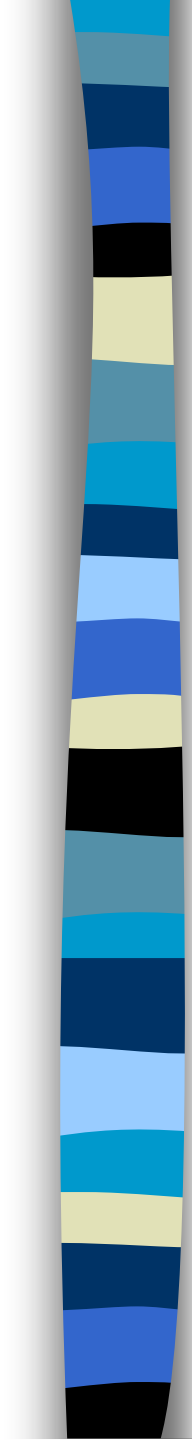
After input I, the lower limit for the new interval is:

$\text{Previous}_{\text{low}} + \text{Range} \times \text{subinterval}_{\text{low}}$ of symbol I.

the upper limit for the new interval is:

$\text{Previous}_{\text{low}} + \text{Range} \times \text{subinterval}_{\text{high}}$ of symbol I.





Previous_{low} = 0, Previous_{high} = 1

Range = 1 - 0 = 1

subinterval_{low} of $l = 0.05$

subinterval_{high} of $l = 0.25$

New interval_{low} = $0 + 1 \times 0.05 = 0.05$

New interval_{high} = $0 + 1 \times 0.25 = 0.25$

After input l ,

$[0, 1) \rightarrow [0.05, 0.25)$

3. After the 2nd l , we have

New interval_{low} = $0.05 + [0.25 - 0.05] \times 0.05 = 0.06$

New interval_{high} = $0.05 + [0.25 - 0.05] \times 0.25 = 0.10$



4. For the 3rd input u, we have

$$\text{New interval}_{\text{low}} = 0.06 + [0.1-0.06] \times 0.25 = 0.07$$

$$\text{New interval}_{\text{high}} = 0.06 + [0.1-0.06] \times 0.35 = 0.074$$

5. The calculations described in previous steps are repeated using the limits of the previous interval and subinterval ranges of the current symbol. This yields the limits for the new interval. After the symbol?, the final range is [0.0713336, 0.0713360)

6. There is no need to transmit both values of the bounds in the **last interval**. Instead, we transmit a value that is **within the final range**. In the above example, any number such as 0.0713336, 0.0713334, ..., 0.0713355 could be used.

if 0.0713348389 is used

$$= 2^{-4} + 2^{-7} + 2^{-10} + 2^{-15} + 2^{-16}$$

→ 16 bits are required

Arithmetic coding yields better compression because it **encodes a message as a whole new symbol instead of separate symbols**



■ Decoding Process

- Given the symbol probabilities, to each symbol in our alphabet we assign a unique number (i) and we associate a cumulative probability value cumprob_i .

S_i	i	Cumprob_i
K	7	0.00
l	6	0.05
u	5	0.25
w	4	0.35
e	3	0.40
r	2	0.70
?	1	0.90
	0	1.00

Given the **fractional representation** of the input codeword value, the following algorithm outputs the corresponding decoded message.



■ DecodeSymbol (value):

Begin

Previous_{low} = 0

Previous_{high} = 1

Range = Previous_{high} - Previous_{low}

Repeat

Find i such that

$$Cumprob_i \leq \frac{value - Previous_{low}}{Range} < Cumprob_{i-1}$$

Output symbol corresponding to i from the decoding table.

Update:

Previous_{high} = Previous_{low} + Range x Cumprob_{i-1}

Previous_{low} = Previous_{low} + Range x Cumprob_i

Range = Previous_{high} - Previous_{low}

Until symbol decoded is ?

End



■ Ex:

1. Initially, $\text{Previous}_{\text{low}}=0$, $\text{Previous}_{\text{high}}=1$, and $\text{Range}=1$.

For $i=6$,

$$\text{Cumprob}_i \leq \frac{\text{value} - \text{Previous}_{\text{low}}}{\text{Range}} < \text{Cumprob}_{i-1}$$

Thus, the first decoded symbol is l.

Update:

$\text{Previous}_{\text{high}}$	$=$	0.25
$\text{Previous}_{\text{low}}$	$=$	0.25
Range	$=$	0.20

2. We repeat the decoding process and find that $i=6$ satisfies again the limits

$$\text{Cumprob}_i \leq \frac{\text{value} - 0.05}{0.20} < \text{Cumprob}_{i-1}$$

Thus the 2nd decoded symbol is l.

Update:

$\text{Previous}_{\text{high}}$	$=$	0.10
$\text{Previous}_{\text{low}}$	$=$	0.06
Range	$=$	0.04



3. Repeating the decoding process yields the $i=5$ satisfies the limits

$$Cumprob_i \leq \frac{value - 0.06}{0.04} < Cumprob_{i-1}$$

Thus the 3rd decoded symbol is u

Update: Previous_{high} = 0.074
Previous_{low} = 0.070
Range = 0.004

4. Repeat the decoding process **until ? is decoded**, then terminate the decoding algorithm,



Implementation Issues :

■ Incremental Output

The encoding method we have described generates a compressed bit stream only after reading the entire message. However, in most implementations, and particularly in image compression, we desire an **incremental transmission scheme**.

From the encoding figure, we observe that after encoding u , the subinterval range is $[0.07, 0.074)$. In fact, it will start with the value 0.07 ; hence we can transmit the first two digits 07 . After the encoding of the next symbol, the final representation will begin with 0.071 since both the upper and the lower limits of this range contain 0.071 . Thus, we can transmit the digit 1 . We repeat this process for the remaining symbols. Thus, incremental encoding is achieved by transmitting to the decoder each digit in the final representation as soon as it is known. The decoder can perform incremental calculation too.

: encoding / decoding details can be found in

“我把電腦變大了” (黃鶴超)

‘數字編碼篇’



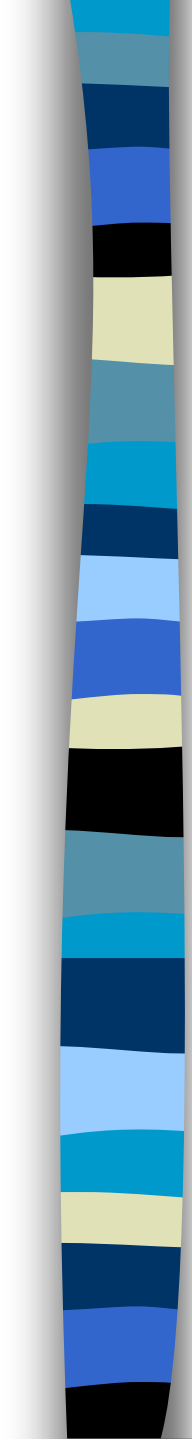
■ High-precision arithmetic

Most of the computations in arithmetic coding use floating-point arithmetic; however, most low-cost hardware implementations support only fixed-point arithmetic. Furthermore, division (used by the decoder) is undesirable in most implementations.

Consider the problem of arithmetic precision in the encoder.

During encoding, the subinterval range is narrowed as each new symbol is processed. Depending on the symbol probabilities, the precision required to represent this range may grow; thus, there is a potential for overflow or underflow. For instance, in an integer implementation, if the fractional values are not scaled appropriately, different symbols may yield the same limits of $\text{Previous}_{\text{high}}$ and $\text{Previous}_{\text{low}}$; that is, no subdivision of the previous subinterval takes place. At this point, encoding would have to be abnormally terminated.





Let subinterval limits $\text{Previous}_{\text{low}}$ and $\text{Previous}_{\text{high}}$ be represented as integers with C bits of precision. The length of a subinterval is equal to the product of the probabilities of the individual events. If we represent this probability with f bits of precision to avoid overflow or underflow, we required $f \leq c+2$ and $f+c \leq p$, where p is the arithmetic precision for the computations.

For AR coding on a 16-bit computer, $p=16$. If $c=9$, then $f=7$. If the message is composed of symbols from a k -symbol alphabet, then $2^f \geq k$. Thus, a 256-symbol alphabet cannot be correctly encoded using 16-bit arithmetic.





- Probability Modeling

Thus far, we have assumed a priori knowledge of the symbol probabilities p_i .

In many practical implementations that use the arithmetic coder, **symbol probabilities are estimated as the pixels are processed**. This allows the coder to **adapt better to changes in the input stream**.

A typical example is a document that includes **both text and images**. Text and images have quite **different probability symbols**. In this case, an adaptive arithmetic coder is expected to perform better than a nonadaptive entropy coder.



good references :

1. Witten, Neal, Cleary, "Arithmetic coding for data compression," Communication ACM, 30(6), pp.520-540, June 1987.
2. Mitchell & Pennebaker, "Optimal hardware and software arithmetic coding procedures for the Q-coder," IBM J. of Research and Development, 32(6), pp. 727-736, Nov. 1988.
3. H.C. Huang, & Ja-Ling Wu, "Windowed Huffman coding algorithm with size adaptation," IEE Proceeding-I, pp. 109-113, April 1993.
4. Chia-Lun Yu & Ja-Ling Wu, "Hierarchical dictionary model and dictionary management policies for data compression," pp. 149-155, Signal Processing, 1998.

