• Arithmetic code

- Why not use Huffman code?
  Good: It is optimal. i.e. shortest possible code.

Bad:
1. Bad for skewed probability
2. Employ block coding helps, i.e. use all N symbols in a block. The redundancy is now 1/N. However,

* Must re-compute the entire table of block symbols if the probability of a single symbol changes.
* For N symbols, a table of $|X|^N$ must be pre-calculated to build the tree.
* Symbol decoding must wait until an entire block is received.

- No need for pre-calculated tables of big size
- Easy to adapt to change in probability
- Single symbol can be decoded immediately without waiting for the entire block of symbols to be received.
- Basic idea in arithmetic coding

* Sort the symbols in lexical order
* represent each string x of length n by a unique interval $[F(x_{i-1}), F(x_i))$ in [0,1). $F(x_i)$ is the cumulative distribution.
* $F(x_i) - F(x_{i-1})$ represents the probability of $x_i$ occurring. (by definition)
* The interval $[F(x_{i-1}), F(x_i)]$ can itself be represented by any number, called a tag, within the half open interval.
* The $l(x) = \lceil \log \frac{1}{P(x)} \rceil + 1$ significant bits of the tag $0.t_1t_2t_3\ldots$ is the code of $x$. That is, $0. t_1t_2t_3\ldots t_k000$ is in the interval $[F(x_{i-1}), F(x_i)]$.

Tag in Arithmetic code

$$T([F(x_{i-1}, x_i))] = \sum_{k=1}^{i-1} P(X = x_k) + \frac{P(X = x_i)}{2} = F_X(X_{i-1}) + \frac{F_X(x_i) - F_X(x_{i-1})}{2}$$

Arithmetic Coding Example

$\{a, b\} \rightarrow \{0, 1\}$, $P(a) = 1/3$, $P(b) = 2/3$
Add the boundary value together then divide 2, we could get the tag.

bba: $15/27, 19/27$
tag$=\frac{15/27+19/27}{2} = 17/27 = 0.101000010\ldots$ (binary value)
l$(x) = \lceil \log \frac{1}{P(x)} \rceil + 1 = 4$, so we take the first 4 bits to get the code: code=1010.

Progressive Decoding

At first we assuming that the probability is half to half, then we move the interval. Each additional bit received narrows down the possible interval. (We do not need any table of the probability at first.)

Uniqueness of the Arithmetic Code

Proof: Know tags within interval and then truncate the tags to get the code. The binary present is prefix the before one, all the interval are within source one. show that:
(1) $F(x_{i-1}) < \lceil T(x_i) \rceil l(x_i) < F(x_i)$
(2) suppose $a = \{b_1, b_2\ldots b_n\}$ (binary), b has a as a prefix then
\( b \in (a, a + \frac{1}{2n}) \) interval (Actually this is a fact)

(3) \( F(x_{i-1}) < \lfloor \overline{T}(x_i) \rfloor l(x_i) + \frac{1}{2n} < F(x_i) \)

(1) \[ \lfloor \overline{T}(x_i) \rfloor l(x_i) - F(x_{i-1}) > 0 \]

\[ \lfloor \overline{T}(x_i) \rfloor l(x_i) - F(x_{i-1}) = \frac{P(x_i)}{2} - (\overline{T}(x_i) - \lfloor \overline{T}(x_i) \rfloor) > \frac{P(x_i)}{2} - \frac{1}{2l(x_i)} \]

(because \( \overline{T}(x_i) - \lfloor \overline{T}(x_i) \rfloor < \frac{1}{2l(x_i)} \))

Now, \( l(x_i) = \lceil log \frac{1}{P(x_i)} \rceil + 1 \Rightarrow log \frac{1}{P(x_i)} = l(x_i) - 1 \)

\( \Rightarrow \frac{1}{P(x_i)} = 2^{l(x_i)-1} \Rightarrow P(x_i) = \frac{2}{2^{l(x_i)}} \)

(2) \( a = 0.b_1b_2b_3...b_n \)

\( b = 0.b_1b_3...b_n b_{n+1}... \)

0.000...1 > b - a (just proof the facts)

(3) \( F(x_i) - \lfloor \overline{T}(x_i) \rfloor - \frac{1}{2^{l(x_i)}} > 0 \)

\( F(x_i) - \lfloor \overline{T}(x_i) \rfloor - \frac{1}{2^{l(x_i)}} > F(x_i) - \overline{T}(x_i) - \frac{1}{2^{l(x_i)}} = \frac{P(x_i)}{2} - \frac{1}{2^{l(x_i)}} = 0 \)

Arithmetic code must be prefix.

- Efficiency of the Arithmetic Code

Off at most by 2/N

\[ \frac{H(X_1,X_2,X_3,...,X_N)}{N} \leq l_A < \frac{H(X_1,X_2,X_3,...,X_N)}{N} + \frac{2}{N} \]

Proof: we have known that

\( H(X) \leq \sum x_i \cdot P(X_i)(\lceil log \frac{1}{P(x_i)} \rceil + 1) \leq H(X) + 2 \)

Because \( X_i \) is a string, not the single symbol. So we divide symbol number in each string and we could get:

\[ \frac{H(X)}{n} \leq \frac{\sum x_i P(X_i)(\lceil log \frac{1}{P(x_i)} \rceil + 1)}{n} \leq \frac{H(X)+2}{n} \]

- Adaptive Arithmetic Code

We assume 1/2 at first, and then update when observation. So we don’t need anything at beginning.

\[ p_n = \frac{1 + \text{count}_{1 \leq i \leq nk}(X_i = b)}{1 + n} \]

\( p_1 = 0.5 \)

\( p_2 = 1/3 \text{or} 2/3 \)

\( p_3 = 1/4 \text{or} 1/2 \text{or} 3/4 \)
\[ p_4 = \ldots \]
Coder and Decoder only need to calculate the probabilities along the path that actually occurs.

- **Dictionary Coding**

Both encoder and decoder are assumed to have the same dictionary (table), so the encoder just sent the indices to decoder and the decoder will know the coding by check the dictionary (table).

- **Ziv-Lempel Coding (ZL or LZ)**

  - Named after J.Ziv and A.Lempel (1977)
  - Adaptive dictionary technique.

  * Store previously coded symbols in a buffer.
  * Search for the current sequence of symbols to code.
  * If found, transmit buffer offset and length.

  - **LZ77**

    Output triplet \( \text{offset}, \text{length}, \text{next} \).

    If the size of the search buffer is \( N \) and the size of the alphabet is \( M \) we need:

    \[
    \lceil \log(N + 1) \rceil + \lceil \log(N + 1) \rceil + \lceil \log M \rceil
    \]

    bits to code a triplet.

  - **Drawbacks of LZ77**

    * Repetitive patterns with a period longer than the search buffer size are not found.
    * If the search buffer size is 4, the sequence
      \( a b c d e a b c d e a b c d e a b c d e \ldots \)
      will be expanded, not compressed. Because that when you
have got \{b\ c\ d\ e\} and the next is a, you could never find the suitable math due to the buffer size which is 4.

• Markov Chain
  - Given 3 random variables X,Y,Z, they form a Markov chain denoted as X→Y→Z (also go other way:X←Y←Z) if
    \[ p(x, y, z) = p(x)p(y|x)p(z|y) \iff p(z|x, y) = p(z|y) \]
  - A Markov chain X→Y→Z means that
    * The only way that X affects Z is through the value of Y
    * \( I(X; Z|Y) = 0 \iff H(Z|Y) = H(Z|X, Y) \) (Given Y, you know every information so \( I(X; Z|Y) = 0 \) )
    * If you already know Y, then observing X gives you no additional information about Z
    * If you know Y, then observing Z gives you no additional information about X
  - A common special case of a Markov chain is when Z=f(Y)
  - Markov Chain Symmetry
    * \( X\rightarrow Y\rightarrow Z \leftrightarrow X\leftarrow Y\leftarrow Z \) (so obvious)
  - Data Processing Theorem
    * If \( X\rightarrow Y\rightarrow Z \) then \( I(X; Y) \geq I(X; Z) \)
      
      • Processing Y cannot add new information about X
    * If \( X\rightarrow Y\rightarrow Z \) then \( I(X; Y) \geq I(X; Y|Z) \)
Knowing Z can only decrease the amount X tells you about Y

* Proof: (1) $I(X;Y) - I(X;Z) = H(X) - H(X|Y) - H(X) + H(X|Z)
  \geq -H(X|Y) + H(X|Z, Y)
  = -H(X|Y) + H(X|Y) = 0$

(2) $I(X;Y) \geq I(X : Y|Z)$

$I(X;Y) - I(X : Y|Z) = H(X) - H(X|Y) - H(X|Z) + H(X|Y, Z)
= H(X) - H(X|Z) = 0$

– Non-Markov: Conditioning Increase Mutual Information

Noisy channel Z=X+Y
$I(X, Y) \geq I(X; Y|Z)$ because $X\rightarrow Y\rightarrow Z$ is not true.
$I(X;Y)=0$ because x&y are independent
Now, $I(X; Y|Z) = H(X|Z) - H(X|Y, Z)$
$(H(X|Y, Z) = 0$ and $H(X|Z) = \sum_i p(z)H(X|Z = Z_i) = 1/2$
so proved