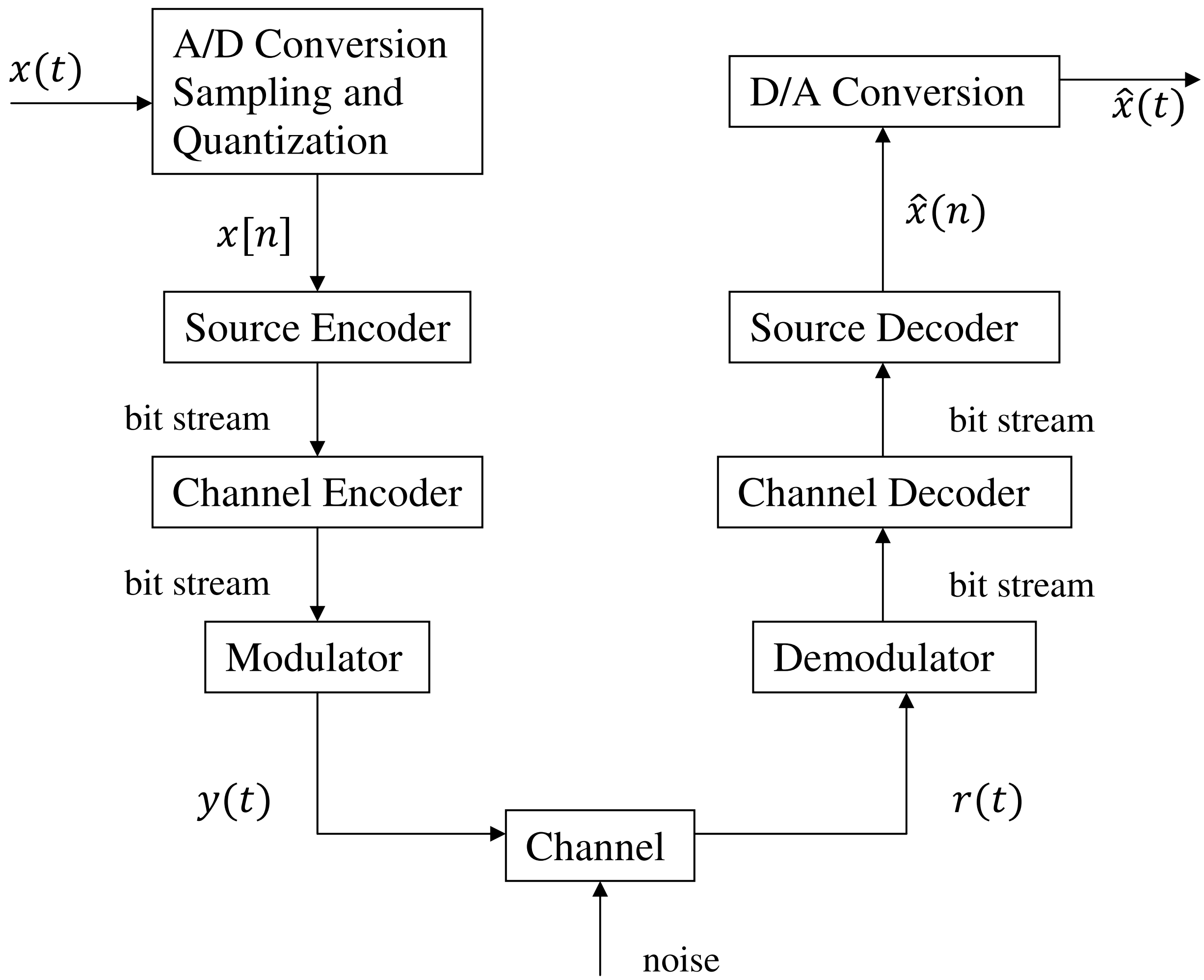
Information Theory and Huffman Coding

- Consider a typical Digital Communication System:



- The "channel" could be a physical communication channel or just a CD, hard disk, etc. in a digital storage system.
- The purpose of a communication system is to convey/transmit messages or information.

Elements of Information Theory

- In 1948, Claude Shannon provided a mathematical theory of communications, now known as **information theory**. This theory forms the foundation of most modern digital communication systems.

- Information theory provides answers to such fundamental questions like:

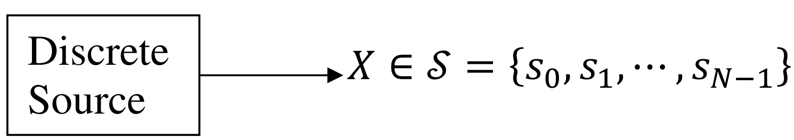
  - What is information --- how to quantify it? What is the irreducible complexity, below which a signal cannot be compressed? (**Source entropy**)

  - What is the ultimate transmission rate (theoretical limit) for **reliable** communication over a **noisy** channel? (**Channel coding theorem**)

- Why digital communication (and not analog), since it involves lot more steps?
  It has the ability to combat noise using channel coding techniques.

- We will consider only the problem of source encoding (and decoding).

- A discrete source (of information) generates one of N possible symbols from a source alphabet set $\mathcal{S} = \{s_0, s_1, \cdots, s_{N-1}\}$, in every unit of time.



  - N is the alphabet size and $\mathcal{S}$ is the set of source symbols.

- **Example**:

- A piece of text in the English language: $\mathcal{S} = \{a, b, \cdots, z\}$; $N = 26$.
- Analog signal $x(t)$, followed by sampling and quantization.

$$x(t) \xrightarrow{\text{sample}} x[n] \xrightarrow{\text{quantize to 8 bits}} \mathcal{S} = \{0, 1, \cdots, 255\}; \; N = 256.$$

- How do we represent each of these symbols $\mathcal{S} = \{s_0, s_1, \cdots, s_{N-1}\}$ for storage/transmission?

- Use a binary encoding of the symbols; i.e., assign a binary string (codeword) to each of the symbols.

- If we use codewords with r bits each, we will have 2^r unique codewords and hence can represent 2^r unique symbols.

- Conversely, if there are N different symbols, we need at least $r = \lceil \log_2(N) \rceil$ bits to represent each symbol.

  - For example, if we have 100 different symbols, we need at least $\lceil \log_2(100) \rceil = \lceil 6.64 \rceil = 7$ bits to represent each symbol. Note that $2^7 = 128 > 100$ but $2^6 = 64 < 100$.

    A possible mapping of the 100 symbols into 7-bit codewords:

Source symbol	Binary codeword
s_0	0000000
s_1	0000001
s_2	0000010
$\vdots$	$\vdots$
s_{99}	1100011

  - For example, if we quantize a signal into 7 different levels, we need $\lceil \log_2(7) \rceil = \lceil 2.807 \rceil = 3$ bits to represent each symbol.

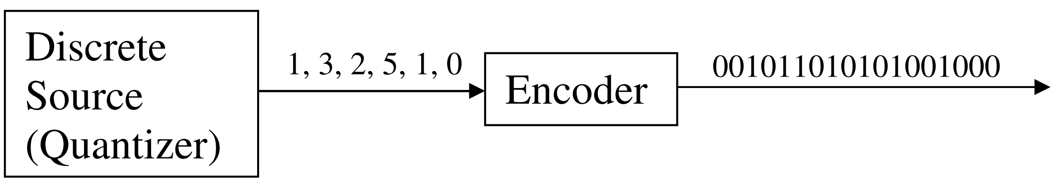
    A possible mapping of the 7 quantized levels into 3-bit codewords:

Symbol	0	1	2	3	4	5	6
Codeword	000	001	010	011	100	101	110

- In both examples above, all codewords are of the same length. Therefore, the average codeword length (per symbol) is 7 bits/symbol and 3 bits/symbol, respectively, in the two cases.

- If we know nothing about the source --- in particular, if we do not know the source statistics --- this is possibly the best we can do.

- An illustration of the encoder for the 7-level quantizer example above:



- A fundamental premise of information theory is that a (discrete) source can be modeled as a probabilistic process.

- The source output can be modeled as a discrete random variable X, which can take values in set $\mathcal{S} = \{s_0, s_1, \cdots, s_{N-1}\}$, with corresponding probabilities $\{p_0, p_1, \cdots, p_{N-1}\}$; i.e., the probability of occurrence of each symbol is given by:
$$P[X = s_n] = p_n, \quad n = 0, 1, \cdots, N-1.$$
Being probabilities, the numbers p_n must satisfy
$$p_n \geq 0 \ \text{ and } \ \sum_{n=0}^{N-1} p_n = 1.$$

- Shannon introduced the idea of "**information gained**" by observing an event $\{X = s_n\}$ as follows:

$$I(s_n) = -\log_2[P\{X = s_n\}] = -\log_2 p_n = \log_2\left(\frac{1}{p_n}\right) \text{ bits.}$$

- The base for the logarithm depends on the units for measuring information. Usually, we use base 2, and the resulting unit for information is "binary digits" or "bits."

- Notice that, each time the source outputs a symbol, the information gain would be different depending on the specific symbol observed.

- The entropy $H(X)$ of a source is defined as the **average information content per source symbol**:

$$H(X) = \sum_{n=0}^{N-1} p_n I(s_n) = -\sum_{n=0}^{N-1} p_n \log_2 p_n = \sum_{n=0}^{N-1} p_n \log_2\left(\frac{1}{p_n}\right) \text{ bits.}$$

- By convention, in the above formula, we set $0 \log 0 = 0$.

- The entropy of a source quantifies the "randomness" of a source. It is also a measure of the rate at which a source produces information.

- Higher the source entropy, more the uncertainty associated with a source output and higher the information associated with the source.

Example:

Consider a coin tossing scenario. Each coin-toss can produce two possible outcomes: Head or Tail denoted as $\{H,\ T\}$.

Note that this is a random source since the outcome of a coin-toss cannot be predicted or known upfront and the outcome will not be the same if we repeat the coin-toss.

Let us consider a few cases:

- **Fair coin**: Here, the two outcomes Head and tail are equally likely. $p_H = p_T = 0.5$. Therefore,
$$I(H) = I(T) = -\log_2 0.5 = -(-1) = 1 \text{ bit.}$$
$$H(X) = p_H I(H) + p_T I(T) = 0.5(1) + 0.5(1) = 1 \text{ bit.}$$

- **Biased coin**: $p_H = 0.9$ and $p_T = 0.1$. Therefore,
$$I(H) = -\log_2 0.9 = 0.152 \text{ bit and } I(T) = -\log_2 0.1 = 3.32 \text{ bit}$$
$$H(X) = 0.9(0.152) + 0.1(3.32) = 0.469 \text{ bit.}$$

- **Very Biased coin**: $p_H = 0.99$ and $p_T = 0.01$. Therefore,
$$I(H) = -\log_2 0.99 = 0.0145 \text{ bit and } I(T) = -\log_2 0.01 = 6.64 \text{ bit}$$
$$H(X) = 0.99(0.0145) + 0.01(6.64) = 0.081 \text{ bit.}$$

- **Extremely Biased coin**: $p_H = 0.999$ and $p_T = 0.001$.
Exercise for you

Example:

Consider the previous 7-level quantizer, where the probabilities of the different levels are as follows:

Symbol s_n	Probability p_n	Information (in bits) $I(s_n) = -\log_2 p_n$
0	$1/2$	1
1	$1/4$	2
2	$1/8$	3
3	$1/16$	4
4	$1/32$	5
5	$1/64$	6
6	$1/64$	6

Source entropy:

$$H(X) = -\sum_{n=0}^{N-1} p_n \log_2 p_n$$

$$= -\left[\frac{1}{2}\log_2\frac{1}{2} + \frac{1}{4}\log_2\frac{1}{4} + \cdots + \frac{1}{64}\log_2\frac{1}{64}\right]$$

$$= \left[\frac{1}{2} + \frac{1}{2} + \frac{3}{8} + \frac{1}{4} + \frac{5}{32} + \frac{3}{32} + \frac{3}{32}\right] = \frac{63}{32} = 1.96875 \text{ bit.}$$

What is the significance of entropy?

- For our source X, all the symbols in $\{0, 1, \cdots, 6\}$ are not equally likely (equiprobable). We may therefore use a variable length code which assigns fewer bits (shorter codeword) to encode symbols with larger probability (e.g., symbol 0, since $p_0 = \frac{1}{2}$) and more bits (longer codeword) to encode symbols with smaller probability (e.g., symbol 6 since $p_6 = \frac{1}{64}$).

- Suppose
  l_0 = # bits used to encode 0, l_1 = # bits used to encode 1, …,
  l_6 = # bits used to encode 6.

- Then average codeword length is defined as:

$$\bar{l} = \sum_{n=0}^{N-1} l_n p_n$$

  and variance of codeword length is defined as:

$$\sigma^2 = \sum_{n=0}^{N-1} p_n \left(l_n - \bar{l}\right)^2$$

- For a fixed length code, we saw earlier that

$$l_n = 3, n = 0, 1, \cdots 6 \ \Rightarrow \bar{l} = \sum_{n=0}^{N-1} 3p_n = 3 \sum_{n=0}^{N-1} p_n = 3$$

  and consequently $\sigma^2 = 0$.

- For a given source, what is the least $\bar{l}$ we can get, using a variable length code?

Prefix-free code

- Note that, if we have a variable length code, it must be uniquely decodable; i.e., the original source sequence must be recoverable from the binary bit stream.
- Consider a source producing three symbols $S = \{a, b, c\}$. Suppose we use the following binary encoding:

Symbol	*a*	*b*	*c*
Codeword	0	1	01

  If we receive a bit stream, say "010" --- it may correspond to source symbols "*aba*" or "*ca*"

  Hence, this is not uniquely decodable (and hence not of any use).

- One way to ensure that a code is uniquely decodable is to have it satisfy the so-called prefix-free condition.
- A code is said to be **prefix-free** if no codeword is the prefix (initial part) of any other codeword.
- **Example 1**:

Symbol	*a*	*b*	*c*
Codeword	0	1	01

  Codeword "0" is a prefix of codeword "01." So this code does not satisfy the prefix-free condition. The above code is **NOT a prefix-free code**.

- **Example 2**:

Symbol	*a*	*b*	*c*
Codeword	0	10	11

This code satisfies the prefix condition. It is a **prefix-free code**

- **Result:** A prefix-free code is uniquely decodable.
- Prefix-free codes are also referred to as instantaneous codes.
- We will study an important prefix-free code called the Huffman code.

Huffman Code

- The algorithm is best illustrated by means of an example.
- Consider a source which generates one of five possible symbols $\mathcal{S} = \{a, b, c, d, e\}$. The symbols occur with corresponding probabilities $\{0.2, 0.4, 0.05, 0.1, 0.25\}$.
- Arrange the symbols in descending order of their probability of occurrence.
- Successively reduce the number of source symbols by replacing the two symbols having least probability, with a "compound symbol." This way, the number of source symbols is reduced by one at each stage.
- The compound symbol is placed at an appropriate location in the next stage, so that the probabilities are again in descending order. Break ties using any arbitrary but consistent rule.
- Code each reduced source, starting with the smallest source and working backwards.
- Illustration of the above steps:

Symbol	Prob.	1	2	3	Codeword
b	0.4	0.4	0.4	0.6	1
e	0.25	0.25	0.35	0.4	01
a	0.2	0.2	0.25		000
d	0.1	0.15			0010
c	0.05				0011

Code Assignment:

Symbol	Prob. (p_n)	Codeword	Length (l_n)
a	0.2	000	3
b	0.4	1	1
c	0.05	0011	4
d	0.1	0010	4
e	0.25	01	2

- Average codeword length of the code we designed:

$$\bar{l} = \sum_{n=0}^{4} l_n p_n = 0.2(3) + 0.4(1) + 0.05(4) + 0.1(4) + 0.25(2)$$
$$= 2.1 \text{ bit/symbol.}$$

- Compare this with entropy of the source for which we designed the code

$$H(X) = -\sum_{n=0}^{N-1} p_n \log_2 p_n$$
$$= -[0.2 \log_2 0.2 + 0.4 \log_2 0.4 + 0.1 \log_2 0.1 + 0.05 \log_2 0.05 + 0.25 \log_2 0.25] = 2.04 \text{ bit} < \bar{l}.$$

- Compare this with a fixed length encoding scheme, where we would require $\lceil \log_2(5) \rceil = \lceil 2.32 \rceil = 3$ bit/symbol.

- The resulting code is called a Huffman code. It has many interesting properties. In particular, it is a prefix-free code (no codeword is the prefix of any other codeword) and hence uniquely decodable.

- **Conclusion**: If the symbols are not equiprobable, a (variable length) Huffman code would in general result in a smaller $\bar{l}$ than a fixed length code.

Decoding

- For the source in the previous example, consider a symbol sequence, and its encoding using the Huffman code we designed:

$$adecbaae \rightarrow 00000100100111000000001$$

- How do we decode this binary string using the Huffman code table?

Symb.	Codeword
a	000
b	1
c	0011
d	0010
e	01

$$\mathbf{000}00100100111000000001 \rightarrow a\mathbf{0010}0100111000000001$$
$$\rightarrow ad\mathbf{01}00111000000001 \rightarrow ade\mathbf{0011}1000000001$$
$$\rightarrow adec\mathbf{1}000000001 \rightarrow adecb\mathbf{000}0000$$
$$\rightarrow adecba\mathbf{000}01 \rightarrow adecbaa\mathbf{01} \rightarrow adecbaae$$

- **Exercise**: Decode the binary string 1000001010010000010

Huffman coding example (with ties)

- While arranging the symbols in descending order, one often encounters ties (symbols with the same probability). This is particularly true when the probability of a combined symbol is equal to that of an original symbol.
- In general, ties are broken with a consistent rule. Two common rules to deal with ties are illustrated below.

<u>Combined symbol placed as low as possible</u>

Symbol	Prob. p_n	1	2	3	Codeword	l_n
a	0.4	0.4	0.4	0.6	1	1
b	0.2	0.2	0.4	0.4	01	2
c	0.2	0.2	0.2		000	3
d	0.15	0.2			0010	4
e	0.05				0011	4

Average codeword length of the code we designed:

$$\bar{l}_1 = \sum_{n=0}^{4} l_n p_n = 0.4(1) + 0.2(2) + 0.2(3) + 0.15(4) + 0.05(4)$$

$$= 2.2 \text{ bit/symbol.}$$

Variance of codeword lengths:

$$\sigma_1^2 = \sum_{n=0}^{4} \left(l_n - \bar{l}_1\right)^2 p_n$$

$$= 0.4(1-2.2)^2 + 0.2(2-2.2)^2 + 0.2(3-2.2)^2 + 0.15(4-2.2)^2 + 0.05(4-2.2)^2 = 1.36.$$

Combined symbol placed as high as possible

Symbol	Prob. p_n	1	2	3	Codeword	l_n
a	0.4	0.4	0.4	0.6	00	2
b	0.2	0.2	0.4	0.4	10	2
c	0.2	0.2	0.2		11	2
d	0.15	0.2			010	3
e	0.05				011	3

Average codeword length of the code we designed:

$$\bar{l}_2 = \sum_{n=0}^{4} l_n p_n = 0.4(2) + 0.2(2) + 0.2(2) + 0.15(3) + 0.05(3)$$
$$= 2.2 \text{ bit/symbol.}$$

Variance of codeword lengths:

$$\sigma_2^2 = \sum_{n=0}^{4} \left(l_n - \bar{l}_2\right)^2 p_n$$
$$= 0.4(2-2.2)^2 + 0.2(2-2.2)^2 + 0.2(2-2.2)^2$$
$$+ 0.15(3-2.2)^2 + 0.05(3-2.2)^2 = 0.16.$$

- Note that the codes designed by either rule have the same average codeword length $\bar{l}$. However, the first rule results in a larger variance (measure of variability between codeword lengths) than the second rule.

Exercise: Compute the entropy $H(X)$ of the above source and compare with $\bar{l}$.

Shannon's source coding theorem

- What is the smallest $\bar{l}$ that can be achieved for a given source using a variable length code?

Theorem: Let X be a discrete source with entropy $H(X)$. The average codeword length for any **distortionless encoding** of X is bounded by

$$\bar{l} \geq H(X).$$

In other words, no codes exist that can losslessly represent X if the average codeword length $\bar{l} < H(X)$.

Result: In general, the Huffman codes satisfy

$$H(X) \leq \bar{l} < H(X) + 1.$$

Exercise: Verify that the above is true for the previous Huffman code examples.

Note: We can refine this result by using higher order codes, where we encode a sequence of n symbols at a time (instead of one symbol at a time). In this case

$$H(X) \leq \bar{l} < H(X) + \frac{1}{n}.$$