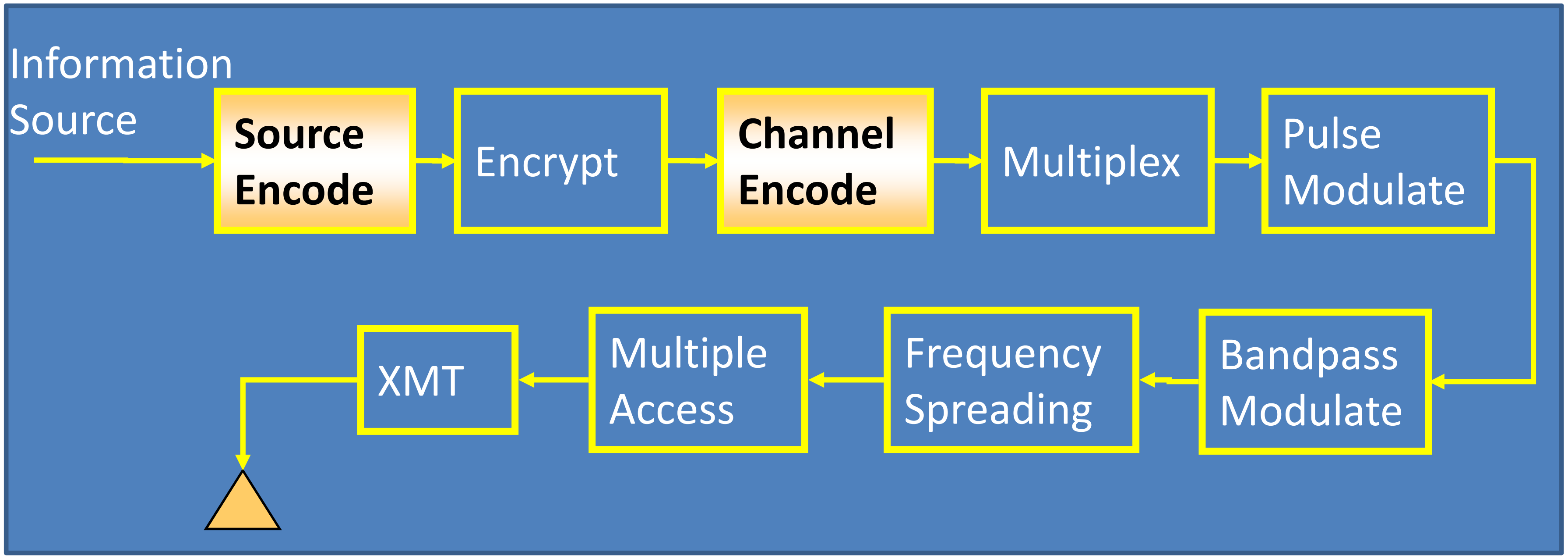
Source Coding

Overview



Source Coding – eliminate redundancy in the data, send same information in fewer bits

Channel Coding – Detect/Correct errors in signaling and improve BER

Source Coding

- Goal is to find an efficient description of information sources
  - Reduce required bandwidth
  - Reduce memory to store
- *Memoryless* –If symbols from source are independent, one symbol does not depend on next
- *Memory* – elements of a sequence depend on one another, e.g. UNIVERSIT_?, 10-tuple contains less information since dependent

$$H(X)_{memory} < H(X)_{no\ memory}$$

Source Coding (II)

$$H(X)_{memory} < H(X)_{no\ memory}$$

- This means that it’s more efficient to code information with memory as groups of symbols

Desirable Properties

- Length
  - Fixed Length – ASCII
  - Variable Length – Morse Code, JPEG
- Uniquely Decodable – allow user to invert mapping to the original
- Prefix-Free – No codeword can be a prefix of any other codeword
- Average Code Length (n_i is code length of *i*th symbol)

$$\bar{n} = \sum_i n_i P(X_i)$$

Uniquely Decodable and Prefix Free Codes

- Uniquely decodable?
  - Not code 1
  - If "10111" sent, is code 3 'babbb'or 'bacb'? Not code 3 or 6
- Prefix-Free
  - Not code 4,
  - prefix contains '1'
- Avg Code Length
  - Code 2: n=2
  - Code 5: n=1.23

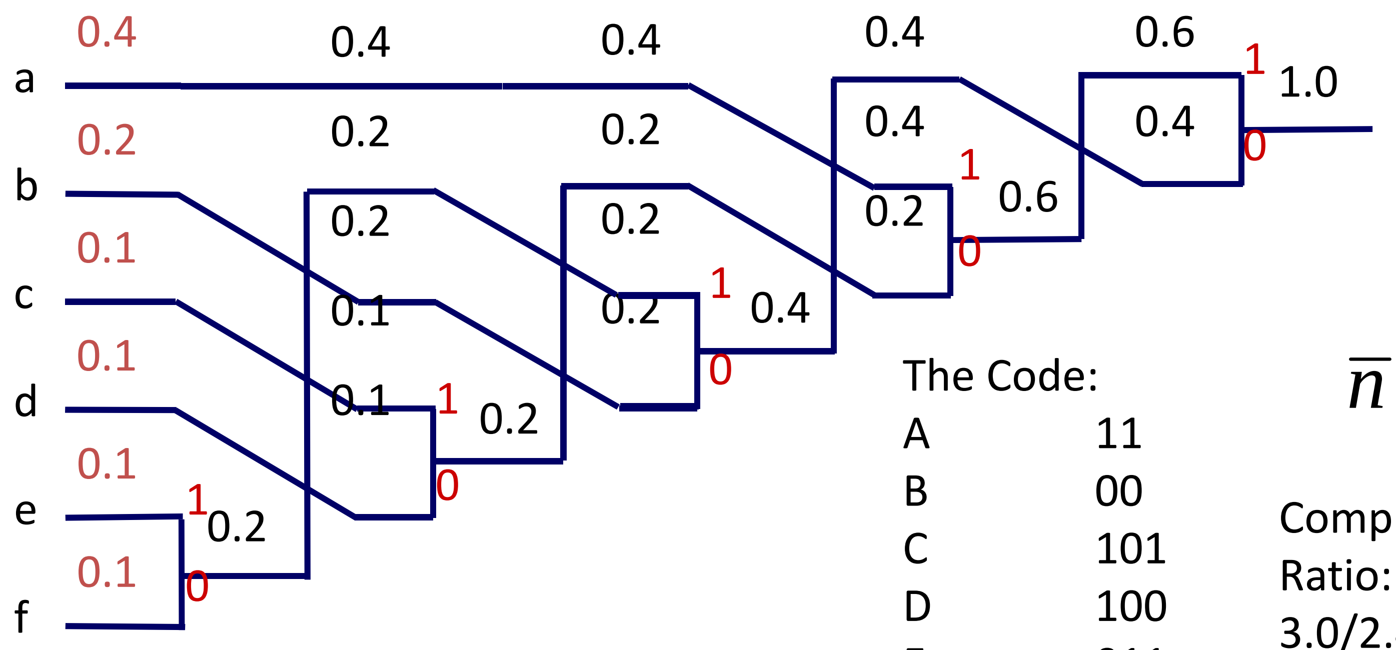
X_i	$P(X_i)$
a	*0.73*
b	*0.25*
c	*0.02*

Symbol	Code 1	Code 2	Code 3	Code 4	Code 5	Code 6
a	*00*	*00*	*0*	*1*	*1*	*1*
b	*00*	*01*	*1*	*10*	*00*	*01*
c	*11*	*10*	*11*	*100*	*01*	*11*

Huffman Code

- Characteristics of Huffman Codes:
  - Prefix-free, variable length code that can achieve the shortest average code length for an alphabet
  - Most frequent symbols have short codes
- Procedure
  - List all symbols and probabilities in descending order
  - Merge branches with two lowest probabilities, combine their probabilities
  - Repeat until one branch is left

Huffman Code Example



The Code:

A	11
B	00
C	101
D	100
E	011
F	010

$$\overline{n} = 2.4$$

Compression Ratio: 3.0/2.4=1.25

Entropy: 2.32

Example:

- Consider a random vector X = {a, b, c} with associated probabilities as listed in the Table

X_i	$P(X_i)$
a	*0.73*
b	*0.25*
c	*0.02*

- Calculate the entropy of this symbol set
- Find the Huffman Code for this symbol set
- Find the compression ratio and efficiency of this code

Extension Codes

- Combine alphabet symbols to increase variability
- Try to combine very common 2,3 letter combinations, e.g.: th,sh, ed, and, the,ing,ion

X_i	$P(X_i)$
aa	0.5329
ab	0.1825
ba	0.1825
bb	0.0625
ac	0.0146
ca	0.0146
bc	0.0050
cb	0.0050
cc	0.0002

Code	n_i	$n_iP(X_i)$
1	1	0.5329
00	2	0.3650
011	3	0.5475
0101	4	0.2500
01000	5	0.0730
010011	6	0.0876
0100100	7	0.0350
01001011	8	0.0400
01001010	8	0.0016

$$\bar{n} = 1.9326\, bits / 2 symbols$$
$$= 0.9663\, bit / symbol$$

Lempel-Ziv (ZIP) Codes

- Huffman codes have some shortcomings
  - Know symbol probability information a priori
  - Coding tree must be known at coder/decoder
- Lempel-Ziv algorithm use text itself to iteratively construct a sequence of variable length code words
- Used in *gzip*, UNIX *compress*, *LZW* algorithms

Lempel-Ziv Algorithm

- Look through a code dictionary with already coded segments
  - If matches segment,
    - send <dictionary address, next character> and store segment + new character in dictionary
  - If no match,
    - store in dictionary, send <0,symbol>

LZ Coding: Example

- Encode [a b a a b a b b b b b b b a b b b b b a]

Code Dictionary

Address	Contents	Encoded Packets
1	a	< 0 , a >
2	b	< 0 , b >
3	aa	< 1 , a >
4	ba	< 2 , a >
5	bb	< 2 , b >
6	bbb	< 5 , b >
7	bba	< 5 , a >
8	bbbb	< 6 , b >
		< 4 , - >

Note: 9 code words, 3 bit address, 1 bit for new character,