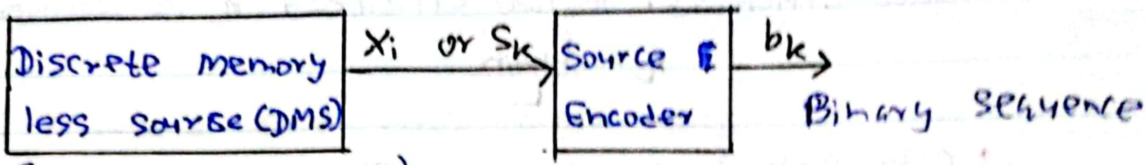
Communication Theory II
Lecture 07
Source Coding Theorem, Huffman Coding

Source Coding



$x = \{x_1, x_2, x_3, \ldots\ldots, x_m\}$

- A conversion of the output of a discrete memoryless source (DMS) into a sequence of binary symbols (i.e, binary code word) is called source coding.
- The device that performs this conversion is called the source encoder. Above figure shows a source encoder.

- There are few terms related to source coding process:
  - i. Code word length
  - ii. Average code word length
  - iii. code efficiency
  - iv. code redundancy

- **Code Word Length:** Let x be a DMS with finite entropy $H(X)$ and an alphabet $\{x_1, \ldots x_m\}$ with corresponding probabilities of occurrence $P(x_i)$ ($i = 1, \ldots, m$).
- Let the binary code word assigned to symbol x_i by the encoder have length n_i, measured in bits.
- The length of a code word is the number of binary digits in the codeword.

• **Average Code Word Length:** The average code word length per source is given by

$$L = \sum_{i=1}^{m} P(x_i)\, n_i$$

• **Code Efficiency:** The code efficiency η is defined as under:

$$\eta = \frac{L_{min}}{L}$$

• **Code Redundancy:** The code redundancy γ is defined as

$$\gamma = 1 - \eta$$

Source Coding Theorem

• The source coding theorem states that for a DMS X, with entropy ~~H~~ $H(X)$, the average code word length L per symbol is bounded as $L \geq H(X)$

• Further, L can be made as close to $H(X)$ as desired for some suitably chosen one.

• Thus, with $L_{min} = H(X)$ the code efficiency can be written as
$$\eta = \frac{H(X)}{L}$$

• **Classification of code:** classification of codes is best illustrate by an example.

1. Fixed-length codes
2. Variable-Length codes
3. Distinct Codes
4. Prefix-Free codes
5. Uniquely Decodable Codes
6. Instantaneous Codes
7. Optimal Codes

CamScanner

Table. Binary Codes

x_i	Code 1	Code 2	Code 3	Code 4	Code 5	Code 6
x_1	00	00	0	0	0	1
x_2	01	01	1	10	01	01
x_3	00	10	00	110	011	001
x_4	11	11	11	111	0111	0001

- Code 1 & Code 2 are fixed-length codes
- Code3, Code4, Code 5 & Code6 are variable-length codes
- If code is distinct, each codeword is distinguishable from other codewords
- If code inputs no codewords can be found by adding code symbols to another codeword is called Prefix-free code
- In prefix-free code, no codewords ~~are~~ is prefix of another codeword
- Code 2, Code 4 & Code 6 are Prefix-Free codes

- Code 3 of the table is ~~not~~ uniquely decodable. For an example, in binary code sequence 1001 may correspond to the source sequence $x_2x_3x_2$ or $x_2x_1x_1x_2$

- Uniquely decodable codes are called as instantaneous codes.
- The instantaneous codes have the property of previously mentioned that no codeword is prefix of another codeword.
- For this reason, prefix free codes are sometimes known as instantaneous codes.

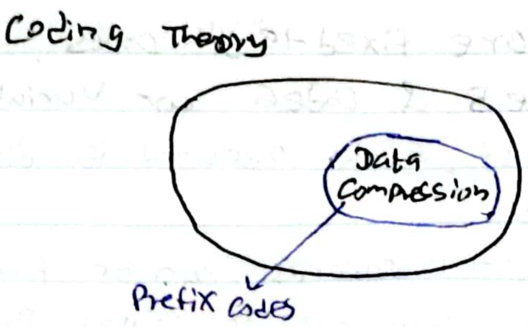
- If code is set to be optimal, if it is instantaneous and has minimum average for a given source, with a given probability for the source symbol

~~Prefix-free codes~~

~~A prefix-free code is one in which no codeword is a prefix of any other. Example~~

Explain prefix codes

Where do prefix codes occur?



Why prefix codes are important

Because

- Unique decodability → This a must have

~~• Inst~~

Imagine

letters	code
a	0
b	1
c	00

decode : 100

It is 'baa' or 'bc'...

Instantaneous decoding → which is a nice-to-have

Imagine

letters	code
a	0
b	01
c	011

decode : 011

1 option: 'c'

we need to keep ~~decoding~~ reading until another zero occurs

Definition

- A prefix code is a code in which no codeword is a prefix (initial segment) to another codeword

letters	Code
a	0
b	10
c	11

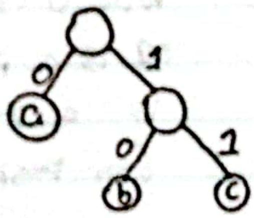
- Prefix codes are also known as:
  - Prefix-free codes
  - Prefix condition codes
  - Instantaneous codes
  - Huffman codes

Letters	Code
a	00
b	01
c	10
d	11

- Fixed-length codes are always prefix-codes

- To check for any code if it is a prefix code, draw its binary tree and check if all codewords are external nodes (i.e. leaves

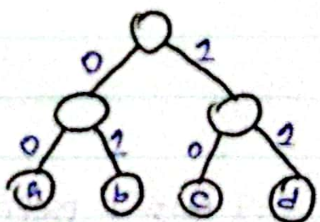
Letters	Code
a	0
b	10
c	11



How to recognize Prefix codes

letters	code
a	00
b	01
c	10
d	11

→

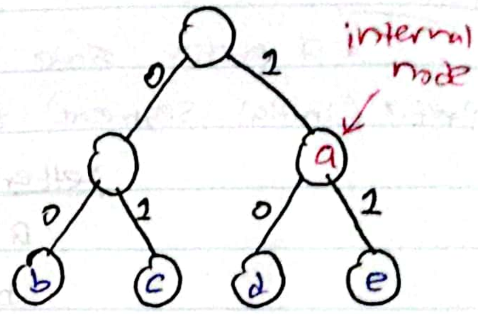


all are external node

Is this a Prefix code?

Yes, since this is a fixed-length code.

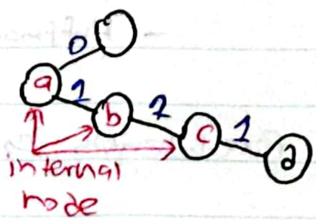
letters	code
a	1
b	00
c	01
d	10
e	11



2. Is this a prefix code?

No, 1 is a prefix to 10 and 11.
(no unique decodability)

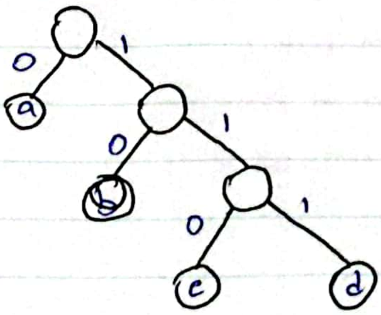
letters	code
a	0
b	01
c	011
d	0111



3. Is this a prefix code?

No, 0 is a prefix to 01, 011 and 0111,
01 is a prefix to 011 and 0111, and 011
is a prefix to 0111
~~(no instantaneous decoding)~~

letters	code
a	0
b	10
c	110
d	1110



All are external nodes

Is this a prefix code?

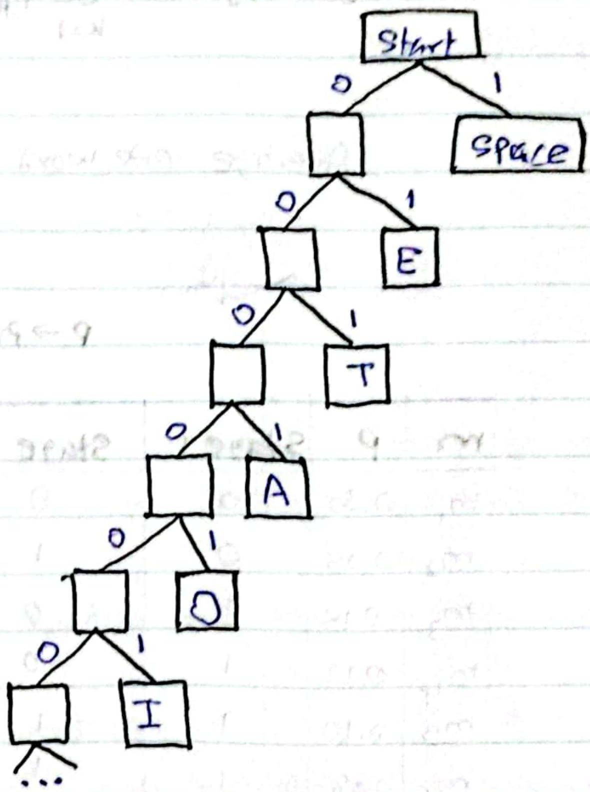
Yes!

Prefix-Free codes [Example 2]

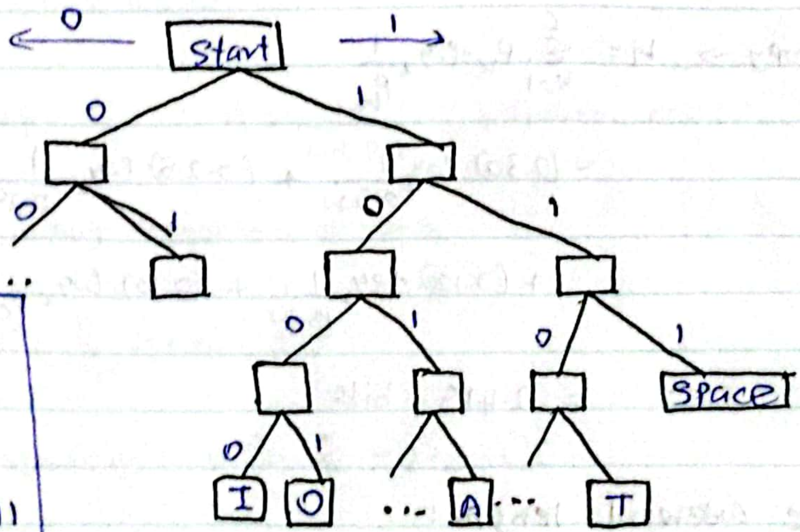
A prefix-free code is one in which no codeword is a prefix of any other

An Example

Space	1
E	01
T	001
A	0001
O	00001
I	000001



I ATE: 00000110001001 01



Space	111
E	010
T	1101
A	1011
O	1001
I	1000

I ATE: 10001111011 1101010

$$\text{Information } I(x_i) = \log_b \frac{1}{p(x_i)} = -\log_b P(x_i)$$

$$\text{Entropy} = \sum_{k=1}^{n} P_k \log_2 \frac{1}{P_k} = -\sum P_k \log_2 P_k$$

Average code word length

$$L = \sum_{k=1}^{n} P_k n_k$$

$P \Rightarrow$ Probability n = number of bits

m	P	Stage 1	Stage 2	Stage 3	codeword	length
m_1	0.30	0	0		00	2
m_2	0.25	0	1		01	2
m_3	0.15	1	0	0	100	3
m_4	0.12	1	0	1	101	3
m_5	0.10	1	1	0	110	3
m_6	0.08	1	1	1	111	3

$$\text{entropy} \rightarrow H = \sum_{k=1}^{6} P_k \log_2 \frac{1}{P_k}$$

$$= (0.30)\log_2 \frac{1}{0.30} + (0.25)\log_2 \frac{1}{0.25} + (0.15)\log_2 \frac{1}{(0.15)}$$

$$+ (0.12)\log_2 \frac{1}{(0.12)} + (0.10)\log_2 \frac{1}{(0.10)} + (0.08)\log_2 \frac{1}{0.08}$$

$$= 2.418 \text{ bits}$$

Average codeword length :

$$L = \sum_{k=1}^{6} P_k n_k$$

$$= (0.30 \times 2) + (0.25 \times 2) + (0.15 \times 3) + (0.12 \times 3) + (0.10 \times 3)$$

$$+ (0.08 \times 3)$$

$$= 2.45 \text{ bits}$$

Efficiency

$$\eta = \frac{L_{min}}{L}$$

If $\eta = 1$ then source code is considered as efficient

$$\eta = \frac{H}{L}$$

Huffman Coding Algorithm

~~Example: Alphabet~~

Steps:

1. The source symbols are arranged in order of decreasing probability. Then the two of lowest probability are assigned bit 0 and 1.

2. Then combine last two symbols and move the combined symbol as high as possible.

3. Repeat the above step until end.

4. Code for each symbol is found by moving backward.

5. eta calculation.

Efficiency $\eta = \frac{H}{L \log_2 r} \rightarrow$ binary $r=2$ (0,1)

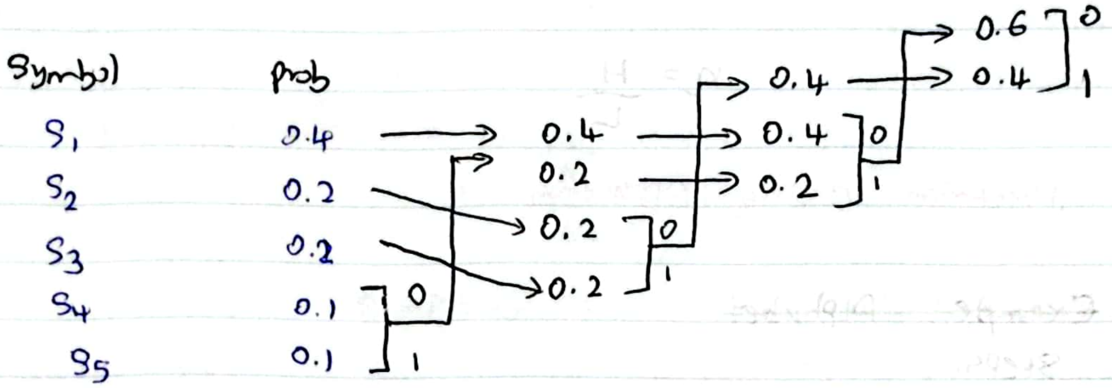
$$L = \text{Avg Codeword} = \sum_{i=1}^{n} P_i n_i$$

$$H = \sum_{i=1}^{n} P_i \log_2\left(\frac{1}{P_i}\right)$$

Variance $\sigma^2 = \sum_{i=1}^{n} P_i (n_i - L)^2$

Example:

Alphabet with prob = $\{0.4, 0.2, 0.2, 0.1, 0.1\}$ for symbols $\{S_1, S_2, \ldots, S_5\}$. Find Huffman codes and also find efficiency & variance.



Symbol	Codeword	length
S_1	00	2
S_2	10	2
S_3	11	2
S_4	010	3
S_5	011	3

- Entropy

$$H = \sum P_i \log_2\left(\frac{1}{P_i}\right)$$
$$= 0.4 \log_2\left(\frac{1}{0.4}\right) + 2 \times 0.2 \log_2\left(\frac{1}{0.2}\right) + 2 \times 0.1 \log_2\left(\frac{1}{0.1}\right)$$
$$= 2.1216 \text{ bits/symbol}$$

- $L = \sum P_i n_i$
$$= 2 \times 0.4 + 2(2 \times 0.2) + 3 \times 0.1 \times 2$$
$$= 2.2 \text{ bits/symbol}$$

$$\eta = \frac{H}{L \log_2 r} = \frac{2.1216}{2.2 \times \log_2 2} = 96.4\%$$

$$\sigma^2 = \sum P_i (n_i - L)^2$$
$$= 0.4(2-2.2)^2 + 2 \times 0.2(2-2.2)^2 + 2 \times 0.1(3-2.2)^2$$
$$= 0.16$$ as low as possible

Huffman Algorithm

Two types of encoding

→ fixed length encoding

↳ assigns to each character a bit string of the same length m

↳ Variable length encoding

↳ assigns codewords of different lengths to different characters

⇒ Huffman algorithm coming under variable length encoding

Huffman Algorithm

~~Each~~ Initialize n one node trees and label them with the characters of the alphabet.

2. Record the frequency of each character in it's tree's root to indicate tree's weight

3. Repeat the following operation until a single tree is obtained. Find two trees with the smallest weight. make them the left and right subtree of a new tree and record the sum of their weights in the root of the new tree as its weight

– A tree constructed by the above algorithm is called a Huffman tree.

~~Ex~~

~~Consider the five character alphabet A, B, C, D, _ with the~~

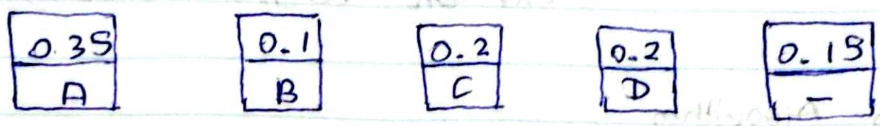
ex

Consider the five character alphabet A, B, C, D, _ with the following occurance probability.

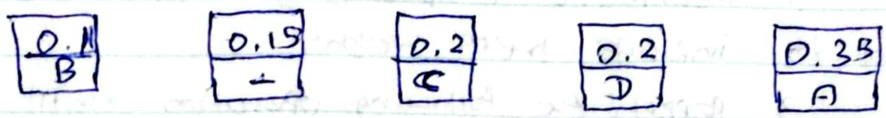
Character	A	B	C	D	_
Probability	0.35	0.1	0.2	0.2	0.15

Step 1

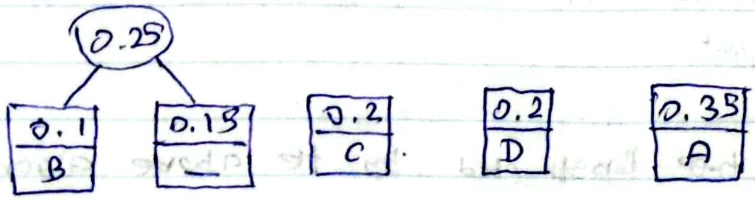
create single node tree



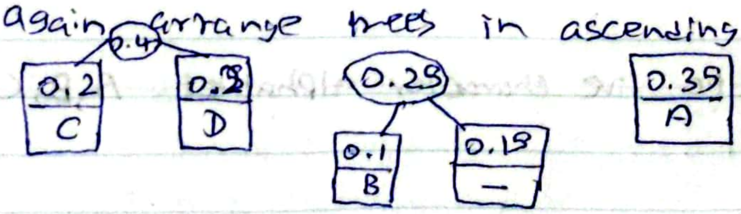
arrange the single node trees with ascending order



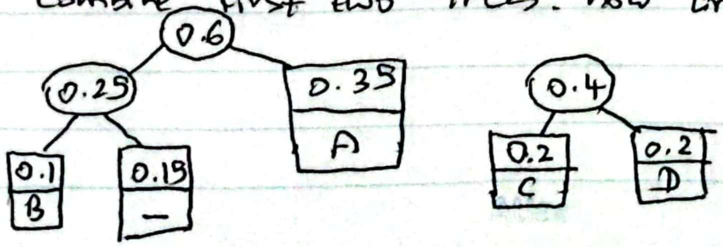
combine first two trees and create new tree



again arrange trees in ascending order

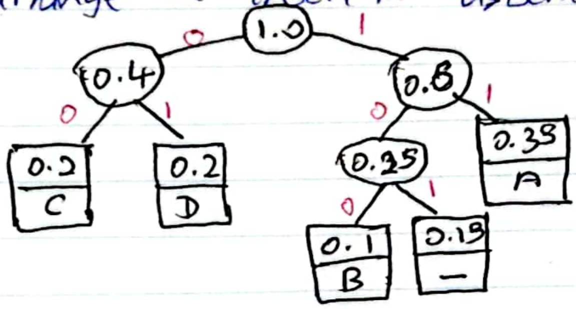


Combine first two trees. now arrange by ascending order



→ combine first two trees. arrange

→ arrange the treen in ascending order



→ now mark as left tree as 0 and right tree as 1

→ now "DAD" is encoded as

01101

→ In the decoding side

DAD