Video Compression

INSTRUCTOR: YAN-TSUNG PENG

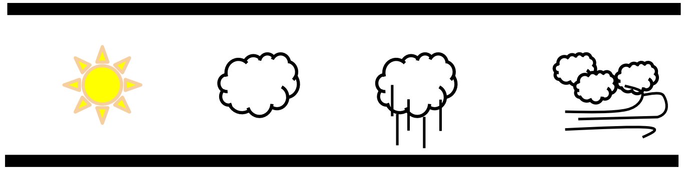
DEPT. OF COMPUTER SCIENCE, NCCU

CLASS 3&4

Entropy Coding

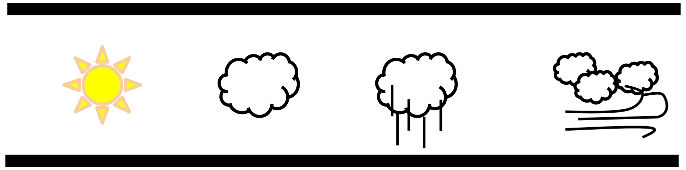
Introduction

Taiwan

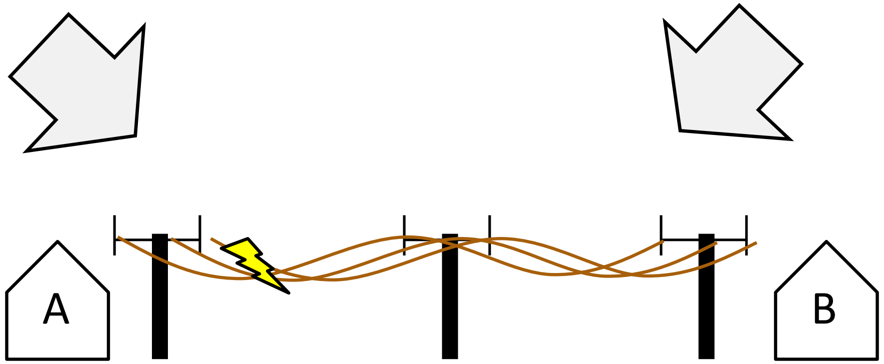


Prob	1/4	1/4	1/4	1/4

Sahara Desert



Prob	1	0	0	0



How many bits do we need to transmit the info?

Examples of “Information”

- I saw a cat that cannot fly
- I saw a flying cat
- I saw a cat chasing a mouse
- I saw a cat riding a motorcycle
- I saw a cat with four feet
- I saw a cat with wings

Information Measurement

- Amount of information carried based on the occurring probability
  - High probability carries little information
  - When a thing occurs rarely – bare a lot of information
- Amount of information carried by an event S is based on the occurring probability p
- $I(S) = \log_2\left(\frac{1}{p}\right) = -\log_2 p$ in bits
  - $p = 1, I(S) = 0$
  - p small, I large
  - $I(S) \geq 0$ for $1 \geq p \geq 0$
  - $I(S_k) > I(S_i)$ for $p_k < p_i$
  - $I(S_k S_i) = I(S_k) + I(S_i)$ if S_k and S_i are independent

Def. of information

Entropy

lossless

- Ideal compressed code length for a symbol ***S*** in lossless compression should be equal to the number of bits that need to be used for information ***S*** has (shortest possible)
- We would like to know the minimum number of bits per symbol required to fully represent the message
- Entropy ***H***
  - Definition: Ideal average bits that need to be used for the information per symbol carries
  - Average compression bits
- Because of overhead of coding, the compressed bits per symbol used are larger than H.
- For lossy compression, the average bits can be smaller than H
- Entropy coding: code length used to represent a symbol is proportional to the amount of the information carried in symbol

Def. of Entropy Coding

frequent symbols : short codes
rare " : long codes

🔍 Example:

Suppose we have 4 symbols with these probabilities:

Symbol	Probability
A	0.4
B	0.3
C	0.2
D	0.1

We calculate **entropy H** like this:

$$H = -\sum p(x)\log_2 p(x)$$

$$= -(0.4\log_2 0.4 + 0.3\log_2 0.3 + 0.2\log_2 0.2 + 0.1\log_2 0.1)$$

$$\approx 1.846 \text{ bits}$$

✅ Meaning:

The **ideal average** number of bits per symbol (in lossless compression) is **~1.846 bits**.
You **can’t do better** than this on average — that’s your **theoretical limit**.

· uncertainty before seeing a message

· information gained after observing it

Entropy Coding

- Entropy of message ***M***: amount of information

$$E(M) = -\sum_i p(m_i)\log_2 p(m_i)$$

Assume we have message m_i with its probability $p(m_i)$

Def. of Shannon Entropy

- For example:
  - Tossing a fair coin. $p(head) = p(tail) = 0.5$
  - Tossing a unfair coin. $p(head) = 0.25 \quad p(tail) = 0.75$

The maximal $E(M)$ for n messages ???

when you have n equally likely messages, $p(m_i) = \frac{1}{n}, \forall i$

$$E(M) = -\sum_{i=1}^{n} \frac{1}{n}\log_2\left(\frac{1}{n}\right) = \log_2 n$$

Common Compression Techniques

- Lossless ◻ Fixed-Length Coding
  - **Run-Length Encoding (RLE):** Encodes consecutive repeated values (runs) as a single value and count, reducing data size.
- **Variable-Length Coding:**
  - **Huffman Coding:** Assigns shorter codes to more frequent symbols and longer codes to less frequent ones, optimizing overall code length.
  - **Arithmetic Coding:** Represents a sequence of symbols as a single number in the interval [0,1), allowing for more efficient compression than Huffman coding in some cases.
- Lossy
  - **Predictive Encoding with Quantization:** Predicts future data points based on past ones and **quantizes** the difference, leading to compression with some loss of fidelity.
    - Intra Coding: Compresses individual blocks or frames independently, as seen in still image compression.
    - Temporal Coding (Motion Estimation): Exploits temporal redundancies between successive frames in video sequences to achieve compression.

Run Length Coding - Lossless

- Run-Length Coding (RLC) is a **simple lossless compression method** that encodes **consecutive repeated values (runs)** as a **single value and count**, reducing data size.

Message: aabbcaaaabbbbbccc

aa bb c aaaa bbbbb ccc

Code:

a2 b2 c1 a4 b5 c3 or (a, 2) (b, 2) (c, 1) (a, 4) (b, 5) (c, 3)

Run: repeated occurrence of the same character

Length: number of repetition

A special character can be used to specify the following character occurs more than 1 time

For example: sssgnnnnn

@s3g@n5 @: special character

sssgnnnnn

Where is Run-Length Coding Used?

- **Image Compression**: Used in **TIFF, BMP**, and **Fax (CCITT Group 3 & 4)** formats.
- **Video Compression**: Applied in **MPEG and Motion JPEG**.
- **Text Compression**: Used in **simple document formats**.

Huffman Coding

- ❑ Shorter code is used for a symbol that occurs more frequent
- ❑ Short code for frequent symbols, long code for rare symbols.
  - ❑ Ex: Morse code
    - ❑ designed according to the frequency of occurrence of each character
    - ❑ the code length is inversely proportional to the frequency
- ❑ Assume the code for symbol s_i has l_i bits, the average code length is:

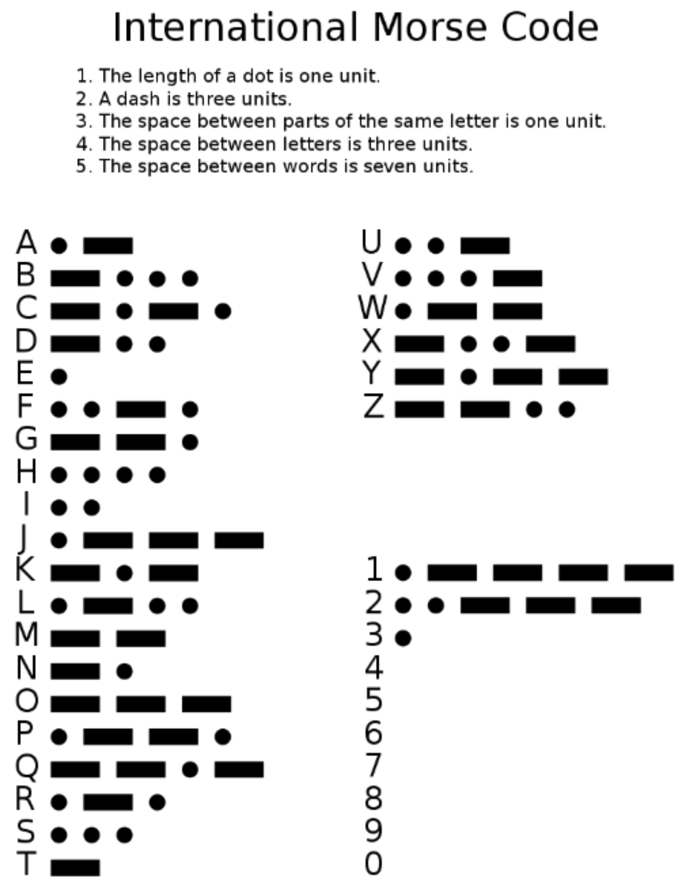
$$L_{avg} = \Sigma_i \, p_i l_i$$

, where p_i is the probability of s_i

- ❑ Unique Prefix Property:
  - ❑ no code is a prefix to any other code
  - ❑ Code must be uniquely decodable:

e.g. $M = \{a, b, c\}$, $C = \{01, 101, 011\}$ (Counterexample)

$\rightarrow$ What is 01101? unclear



Figure's taken from https://en.wikipedia.org/wiki/Morse_code

Shannon's Source Coding Theorem

- Let the entropy of message ***M*** be $E(M)$
  - The average code length for any source encoding technique is bounded as: $L_{avg} \geq E(M)$

theoretical
lower
bound

- The coding efficiency of an encoder is defined as $r_c = \frac{E(M)}{L_{avg}}$

Ex:

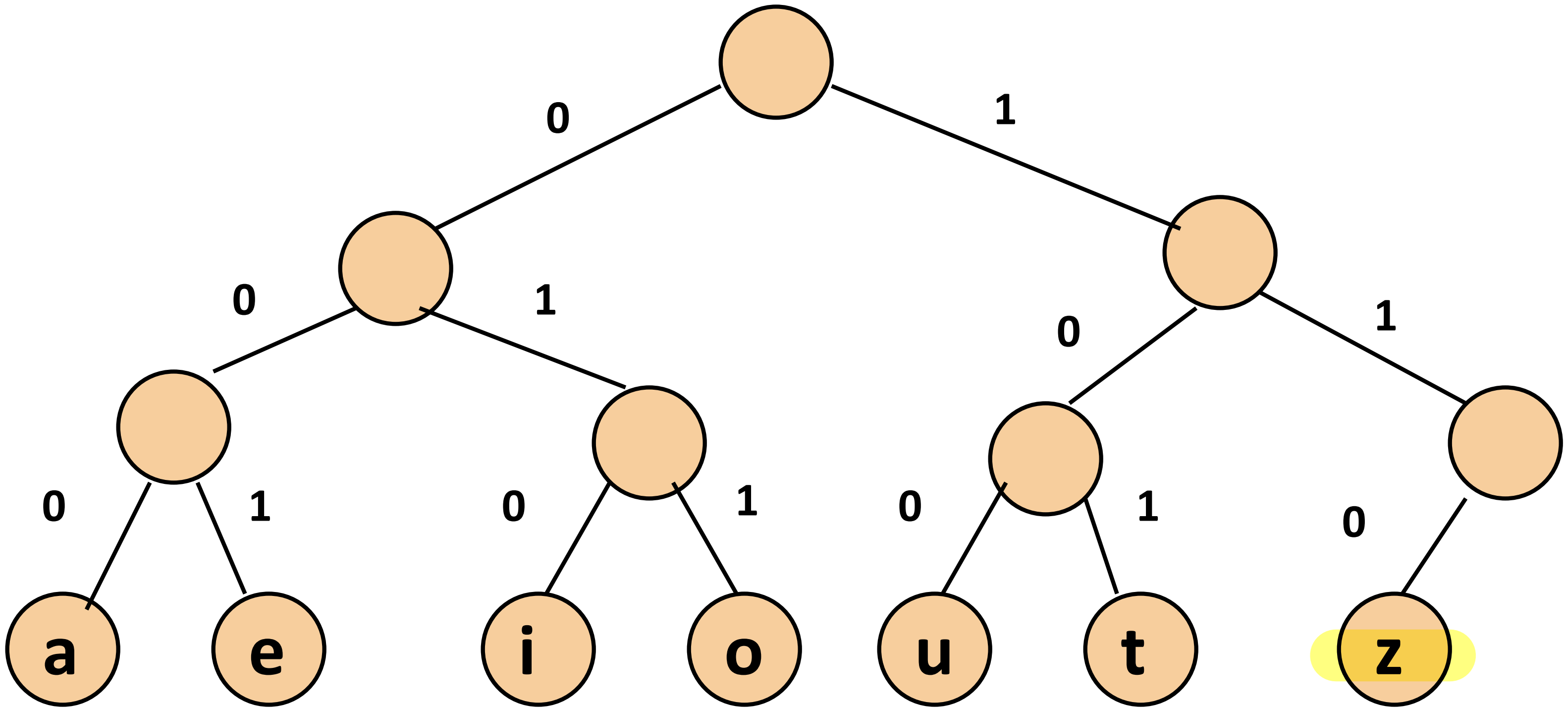
Message (M)	Probability	Code length
A	0.2	20
B	0.4	5
C	0.3	10
D	0.1	30

$$r_c = \frac{-(0.2\log_2 0.2 + 0.4\log_2 0.4 + 0.3\log_2 0.3 + 0.1\log_2 0.1)}{0.2\times 20 + 0.4\times 5 + 0.3\times 10 + 0.1\times 30} \approx 0.154$$

Ref. C.E. Shannon, "A Mathematical Theory of Communication," Bell System technical Journal, Vol 27, pp 379-423 & 623-656, 1948.

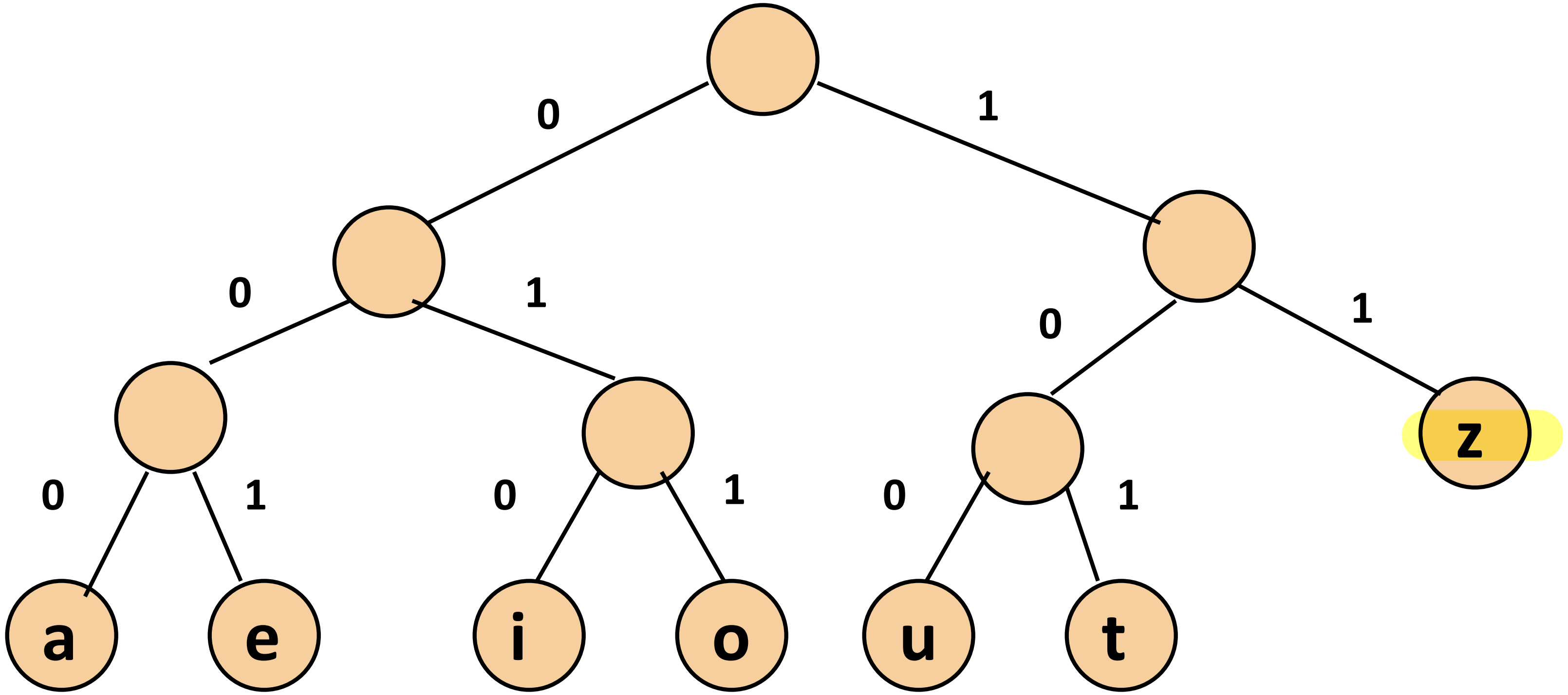
Huffman Coding

- Fixed length coding for a set of 7 symbols



Huffman Coding

- Fixed length coding for a set of 7 symbols



Huffman Coding

- Huffman algorithm
  - greedy strategy
  - Bottom-up approach from the pair with the least probabilities
  - If prior statistics are available, then Huffman coding is near optimal
  - Decoded uniquely

Huffman Coding

Message	Probability
a	11/60
e	16/60
i	12/60
o	13/60
u	3/60
t	4/60
z	1/60

HUFFMANTREE

HuffmanTree (node list L; integer m)
//Each of the m nodes in L has an associated frequency f, and L is
//ordered by increasing frequency; algorithm builds the Huffman tree

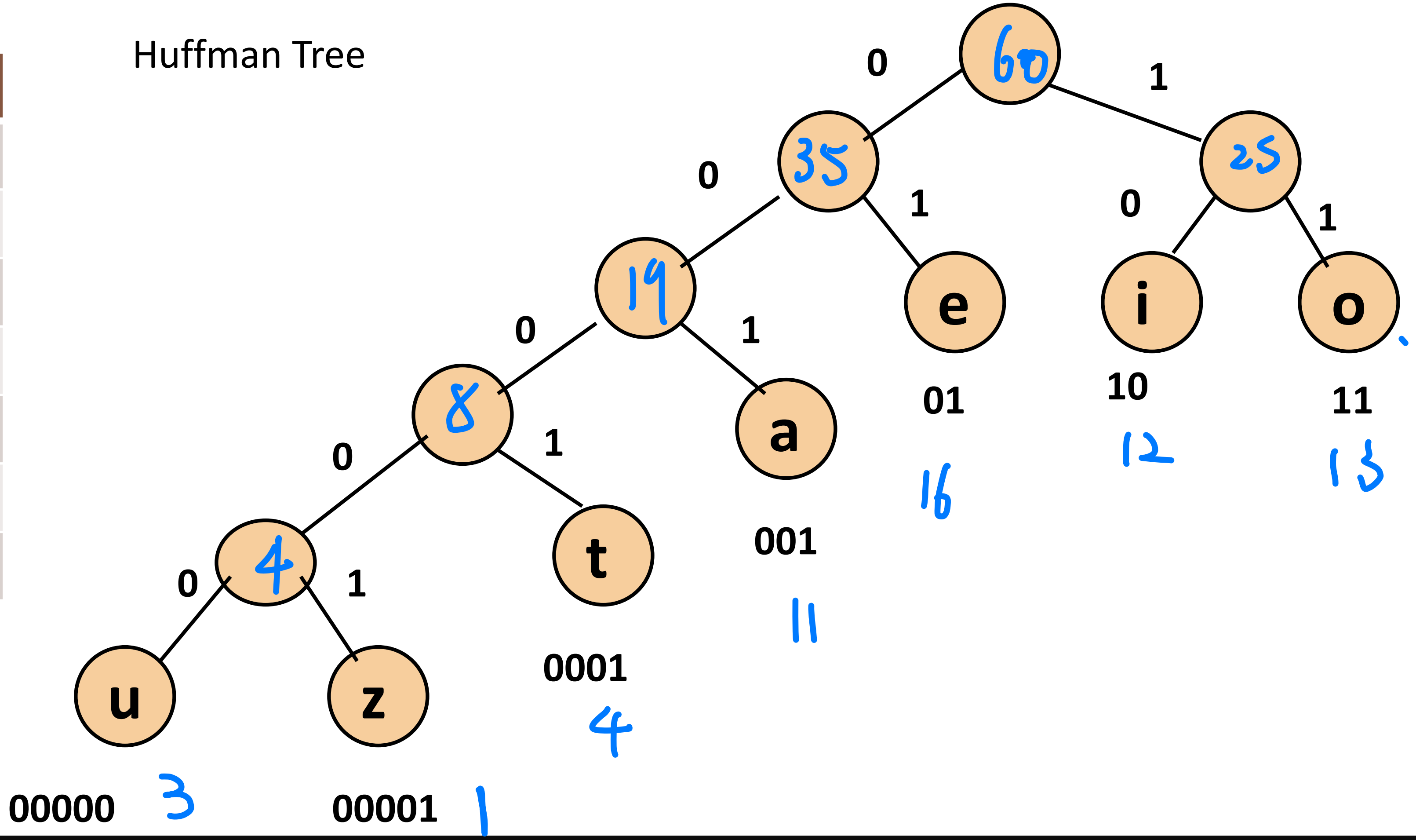
```
for (i = 1 to m − 1) do
    create new node z
    let x, y be the first two nodes in L        //minimum frequency nodes
    f(z) = f(x) + f(y)
    insert z in order into L
    left child of z = node x
    right child of z = node y                   //x and y are no longer in L
end for
end HuffmanTree
```

https://www.cs.mtsu.edu/~xyang/3080/huffman.html

Huffman Coding

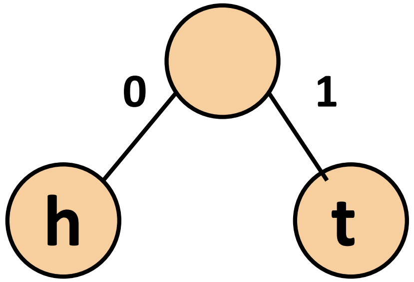
Message	Probability
a	11/60
e	16/60
i	12/60
o	13/60
u	3/60
t	4/60
z	1/60

Huffman Tree



Huffman Coding

- Tossing an unfair coin. $p(head) = 0.25 \quad p(tail) = 0.75$



$$\text{Entropy} = -\sum_i p(m_i)\log_2 p(m_i)$$

$$= -[0.25\log_2(0.25) + 0.75\log_2(0.75)]$$

$$\approx 0.81$$

- Average length = 0.25*1+ 0.75*1=1
- However, its entropy is less than 1

Coding Efficiency $r_c = \frac{Em}{Lavg}$

$\approx 0.81/1 = 0.8$

Limitation of Huffman Coding

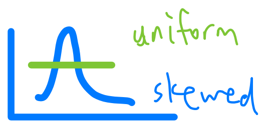
- As known, the lower bound of the average code length L_{avg}^{h} of Huffman coding is the entropy

  Let $E(M) \le L_{avg}^{h} \le E(M) + k$, where $E(M) + k$ is its upper bound.

  It can be proved that k correlates with p_{max},
  where p_{max} represents the largest probability of a symbol that occurs

Thus, if the probability distribution is more balanced and the alphabet (table of symbols) is large, p_{max} will be small, and Huffman coding will perform efficiently to close to the entropy.

If the probability is highly skewed, p_{max} will be large. Then, the coding efficiency of Huffman coding will be lower.



Integer Codeword Length

- Huffman codes always assign integer number of bits to each symbols
  - Tossing a unfair coin. $p(head) = 0.25 \quad p(tail) = 0.75$
  - Using Huffman Coding: The average length L_{avg} = 0.25*1+ 0.75*1=1
  - However, the optimal length should be -0.25$\log_2$(0.25)-0.75$\log_2$(0.75) ≈0.811
- In an extreme case, if $p(head) = 0.9$, it's average codeword length would be 0.152, which is far shorter than 1

Limitation of Huffman Coding

- Huffman coding works inefficiently if the probability tends to be highly skewed.
  - ex: tossing an unfair coin. $p(head)$=0.25 $p(tail)$=0.75
  - solution: creating more longer symbols
    - However, it is more expensive since the Huffman tree MAY grow exponentially as the number of messages increases. (code length grows exponentially for a highly-skewed distribution)
    - Storage size for it would be impractical.

Example : Grouping Symbols

Huffman code for three letters

Symbol (M)	Code	b_n	Prob
A	0	1	0.95
B	11	2	0.02
C	10	2	0.03

skewed

The average length = 1.05 bits/symbol $= 1 \times 0.95 + 2 \times 0.02 + 2 \times 0.03$

Entropy E(M) = 0.074*0.95+5.644*0.02+5.059*0.03 = 0.335

$-\sum p(i) \log p(i)$

Coding Efficiency = 0.335/1.05 ≈ 32%

Coding Efficiency $r_c = \dfrac{\text{average length } L_{avg}}{\text{entropy } E(M)}$

Huffman code for an extended alphabet (group symbols)

↓

flatten probability distribution

Symbol (M)	Code	Prob
AA	0	0.9025
AB	111	0.019
AC	100	0.0285
BA	1101	0.019
BB	110011	0.0004
BC	110001	0.0006
CA	101	0.0285
CB	110010	0.0006
CC	110000	0.0009

The average length = 1.222 bits/symbol

-> for one-letter symbols, it is 0.611 bits/symbol

Coding Efficiency = 0.335/0.611 ≈ 55%

Summary

- Huffman coding works inefficiently if the probability tends to be highly skewed.
  - ex: tossing an unfair coin. $p(head)$=0.25 $p(tail)$=0.75
  - solution: creating more longer symbols
    - However, it is more expensive since the Huffman tree will grow exponentially as the number of messages increase

Message	Probability
a	11/60
e	16/60
i	12/60
o	13/60
u	3/60
t	4/60
z	1/60

Message	Probability
a	10/60
e	14/60
i	11/60
o	10/60
u	3/60
t	4/60
z	1/60
out	1/60
…	…

Golomb Coding

- Lossless Compression
- Invented by Solomon W. Golomb in the 1960s.
- Golomb coding is an optimal prefix coding to messages that conform to a geometric distribution
  - The occurrence of small values in the input stream is significantly more likely than large values.

Geometric Distribution

- It is a type of discrete probability distribution, requiring a number of trials before achieving a successful one, meaning having a certain number of failures before getting the first success.

Under the probability distribution of the number X of Bernoulli trials

- Considering a sequence of Bernoulli trials, where each trial has only two possible outcomes (e.g. failure or success, head or tail, etc.).
- The probability of any one of the outcomes is assumed to be the same for each trial.
- The geometric distribution is used to model the probability of the number of one outcome occurring before the first the other outcome **Failure/Tail vs. Success/Head**

Geometric Distribution

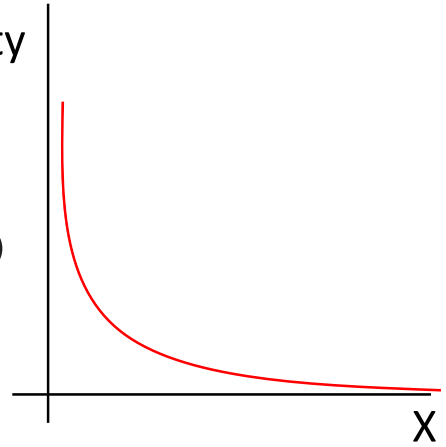
- Example:

Considering rolling a dice repeatedly until a 5 is obtained is a success and getting the other numbers is a failure. Then, the probability of success is $\frac{1}{6}$. Let a random variable, X be the number of trails needed for getting the first success. We say X conforms to a geometric distribution as

geometric distribution

probability

Ex: we need *k* trials to obtain one success (X)



$$P(X = k) = (1 - p)^{k-1}p \quad \text{as } p = \frac{1}{6}$$

This pattern continues (k ↑), showing that **longer sequences before the first success become increasingly unlikely**.

Golomb-Rice Coding

- A subset of the family of Golomb codes
- Generating a simpler (but possibly suboptimal) prefix code
- Golomb code is tunable using any positive integer whereas Rice codes only use a power of two. $m = 2^k$
- Efficient for binary operations (shifts and masks)
- Compared to Huffman Coding, primarily based on the probability from the data, it's based on a simple model of the probability of the values (A smaller value means it is more like to occur)

Golomb, S. W. "" Run-length encodings", IEEE Trans. Inform. Theory, vol. IT12, pp. 399-401." (1966).

Motivation

- Assume we aim to encode 0, 1, 2, … 32.
- Using 5 bits can only encode 0, 1, ..., 31. For example, 00000 is for 0, and 11111 is for 31.
- Thus, we need to use 6 bits to encode this sequence, resulting in a waste of 64-33 = 31 codewords.

$$33 \left\{ \begin{array}{l} \text{000000 for 0} \\ \quad\vdots \\ \text{100000 for 32} \end{array} \right.$$

- If the code conforms to a geometric distribution, where small values have higher probabilities, we can utilize a variable-length coding method to code this sequence.

Unary Code

- Representing a non-negative integer n by
  - n 1’s followed by a 0
  - **<u>n 0’s followed by a 1</u>**
- For example
  - Message – Code
  - 3 – 0001
  - 5 – 000001
  - 0 – 1

Golomb Coding

chosen parameter

$X = q \cdot m + r$

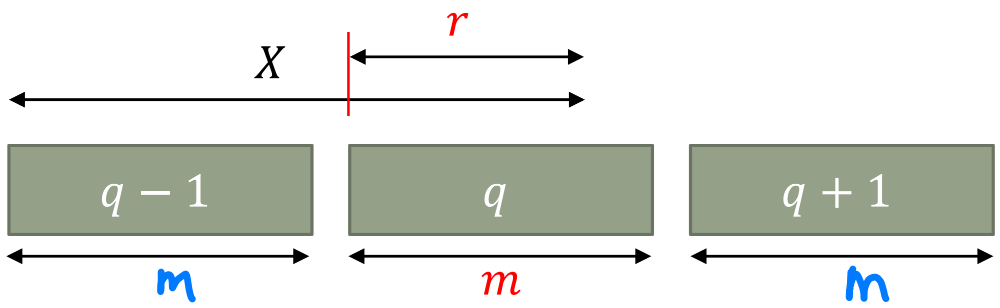
quotient remainder

- ❑ Integer Coding
- ❑ Determine the tunable parameter m as the divisor
  - ❑ chosen based on the probability distribution of values
- ❑ Given a message X, we have
  - ❑ Quotient: $q = \left\lfloor \frac{X}{m} \right\rfloor$ (unary coding)
  - ❑ Remainder: $r = X - qm$ represented as truncated binary code (binary number with certain length)
- ❑ Golomb Coding
  - ❑ **For general Golomb coding:** Use $m = \lceil \frac{\ln(2)}{p} \rceil$ or estimate from **mean values** as $m \approx 2^{\lceil \log_2 E(X) \rceil}$.
    - ❑ Estimate from **mean values:** m is a **power of two**, simplifying encoding and reducing computational costs.
  - ❑ Quotient q : sent as unary code
  - ❑ Remainder r : truncated binary encoding

$X = q \cdot m + r$

Golomb Coding - Representing r

- If m=1, it reduces to unary coding ($q = X$)
- If *m* is a power of two (Rice code)
  - *m* is a pre-defined parameter
  - Represent r with $k = \log_2 m$ bits
- Otherwise (Golomb code)
  - Let $b = \lceil \boldsymbol{log_2\, m} \rceil$
  - For the first $2^b - m$ values, use binary representation with $\boldsymbol{b-1}$ bits (smaller values use shorter code)
  - For the rest values, use binary representation with $\boldsymbol{b}$ bits for representing $r + 2^b - m$



variable length binary encoding (

$m - (2^b - m)$

$= 2m - 2^b$

For unique prefix property

(shift the remaining values)

🔷 Why Use $2^b - m$ and Represent $r + 2^b - m$ in Golomb Coding?

In Golomb coding, when m is not a power of 2, we split an integer X into:

$$X = q \cdot m + r$$

where q is encoded in unary and r is encoded in binary. The challenge is to represent the remainder r efficiently and prefix-free.

🔷 Why Use the First $2^b - m$ Values for Shorter Codes?

We define $b = \lceil \log_2 m \rceil$, giving us 2^b binary codewords of length b. However, we only need to encode m remainders. To avoid wasting code space and to favor smaller, more probable values:

- The first $2^b - m$ values of r are encoded with shorter $b - 1$-bit codes.
- The remaining $m - (2^b - m)$ values are assigned longer b-bit codes.

This structure efficiently fills the available code space while preserving the prefix-free property.

🔷 Why Represent as $r + 2^b - m$?

To avoid gaps and collisions in the binary code space, the remainders beyond the first group are shifted upward. Instead of encoding these r values directly, we encode:

$$r' = r + (2^b - m)$$

This maps them to unused positions in the full b-bit code space, ensuring:

- A continuous, collision-free code assignment
- All m remainders are uniquely encoded
- Prefix-freeness is maintained

✅ Summary

Using $2^b - m$ short codes and shifting the rest by that same amount allows Golomb coding to:

- Encode all remainders efficiently
- Minimize average code length
- Maintain a prefix-free structure using a full binary code tree

Rice Coding - Representing X

- Example
  - $X = 11$ and $m = 4$
  - $k = \log_2 4 = 2$
  - $q = \left\lfloor \frac{11}{4} \right\rfloor = 2$
  - $r = X \bmod m = 3$
  - Unary code for $q = 001$, which is 2
  - r = 11
- Output: 001 11

3 = 0 × 5 + 3

Golomb Coding - Representing X

- Example → possible remainder
  - $X = 3$ and $m = 5$ 0 1 2 | 3 4
  - $q = \left\lfloor \frac{3}{5} \right\rfloor = 0$
  - $r = X - qm = 3$
  - Unary code for $q = 0$, which is 1
  - r = 110
- Final Golomb code for $X = 3$ and $m = 5$
  - q r = 1 110

$b = \lceil \log_2 5 \rceil = 3$

Since m is not a power of two

Encode the remainder using **variable-length binary encoding**

Split the remainder range into **two parts**:

case 1 (i) For the first $2^b - \overset{3}{5}$ values (starting from 0), use binary representation with $\lceil \boldsymbol{log_2}\ \mathbf{5} \rceil$-$\overset{2}{\mathbf{1}}$ bits

- using $b - 1 = 2$ bits to represent 0, 1, 2 (first 3 values)
- 00, 01, 10

case 2 (ii) For the rest values, use binary representation with $b = 3$ bits for representing $r + \boxed{(2^b - 5)}$ r+3

- using 3 bits to represent 3 and 4

(3 + 3) and (4 + 3) with code 110 and 111

Decoding Golomb Code

- Code: 0001 01 with $m = 5$

  0, 1, 2 (2 bits)

  3, 4 → 6, 7 (3 bits)

- Read the bits from the message until the stopping bit is hit.
  - For 000<u>1</u> : $q = 3$ . . . find q

$$b = \lceil \log_2 m \rceil = \lceil \log_2 5 \rceil = 3$$

- For 01

  (i) For 0, 1, and 2, their codes are 00, 01, and 10.

  (ii) For 3 and 4, their codes are 110 and 111.

$$2^b - m = 2^3 - 5 = 3$$

  - So, $r = 1$ . . . find r

- Message $X = qm + r = 16$

  $3 \times 5 + 1$

Golomb-Rice Code

- For q, it is coded using unary coding
- If m is a power of two (Rice coding)
  - Represent r with $\log_2 m$ bits
- If $m = 4$, for numbers up to 15

$r = \log_2 4 = 2$ (bits)

Value	Quotient q	Reminder r	Code
0	0	0	1 00
1	0	1	1 01
2	0	2	1 10
3	0	3	1 11
4	1	0	01 00
5	1	1	01 01
6	1	2	01 10
7	1	3	01 11
8	2	0	001 00
9	2	1	001 01
10	2	2	001 10
11	2	3	001 11
12	3	0	0001 00
13	3	1	0001 01
14	3	2	0001 10
15	3	3	0001 11

Ref: https://urchin.earth.li/~twic/Golomb-Rice_Coding.html?fbclid=IwAR1rjnl5djX32lw9zzqL7GJHcbw1ktC5Sr5GlnQCEwIzK6haAGzFQGvQrKY

Applications : Exp - Golomb Coding

- For coding headers in video coding standards, such as H.265 and H.266

k = len(x+1) - 1 prepend k 0's

- Exp-Golomb coding for Message X
  - Do *X*+1 in binary code with the least length
  - Preceding X+1, add *k* starting zero bits, where
    - *k* = (the length of the binary code of X+1) - 1

Ex.

Encoding X=3

- Compute $X + 1 = 3 + 1 = 4$
- Binary representation of 4 is '100', which has 3 bits.
- Calculate $k = 3 - 1 = 2$
- Prepend 2 zeros to '100', resulting in '00100'.

X	X+1		Code
0	1	0	1
1	10	1	0 10
2	11	1	0 11
3	100	2	00 100
4	101	2	00 101
5	110	2	00 110
6	111	2	00 111
7	1000	3	000 1000
8	1001	3	000 1001
9	1010	3	000 1010

REF: https://en.wikipedia.org/wiki/Exponential-Golomb_coding

Exp-Golomb Decoding

- **Decoding Process:**
  - **Count leading zeros (k):**
    - Determine the number of leading zeros in the codeword.
  - **Extract the following k+1 bits:**
    - These bits represent X+1 binary.
  - **Compute X:**
    - Convert the extracted bits to a decimal number and subtract one: X=binary_to_decimal(bits)−1

Example: Decoding '00100'

1.Count leading zeros: 2.

2.Extract the next 3 bits (2+1): '100'.

3.'100' in binary is 4.

4.Compute $X = 4 - 1 = 3$

Therefore, the decoded value is $X = 3$

Ex. $\underline{000}\,1001$

↳ $3+1=4$

$(1001)_2 = 9$

Arithmetic Coding

- Arithmetic coding is a type of a large block coding method, which can assign code to a longer sequence without generating code for all possible symbols
- A lossless compression method better than Huffman Coding
  - First approached in 1976, by Rissanen from IBM
- It addresses two main issues Huffman coding has:
  - Integer codeword length
  - Hard to make Huffman coding adaptive to the data
- Arithmetic coding is to use the cumulative density function as a hash function to code a long message

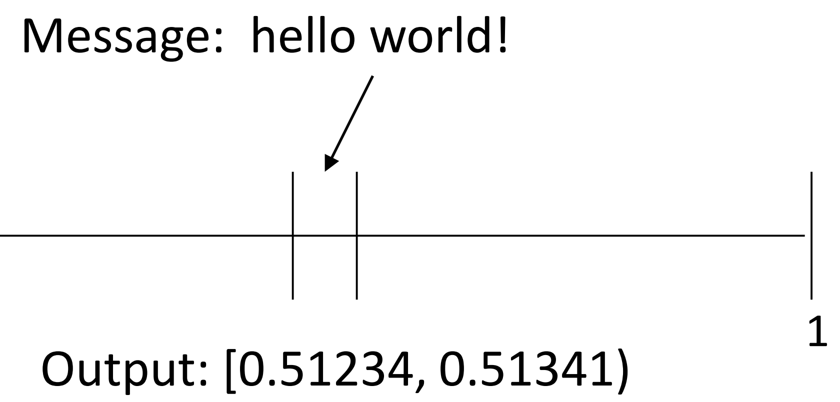
(CDF)

- **A unique code for a sequence with a given length can be solely generated without generating code for all possible sequences of that length.**

Ref.I. H. Witten, R.M. Neal, and J.G. Cleary, "Arithmetic coding for data compression," *Communication of the ACM*, 30, 6(June), 1987, pp. 520-540

Introduction to Arithmetic Coding

- Code each symbol i into a real number in the interval [0, 1)

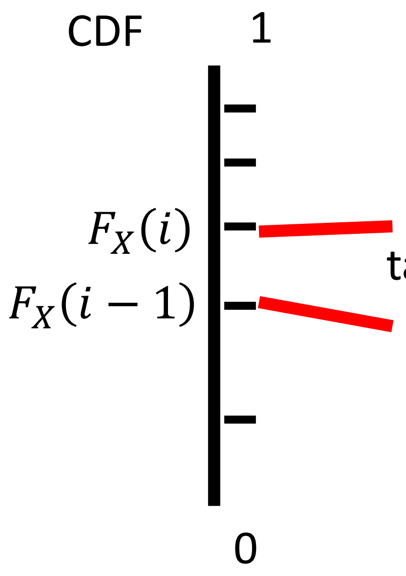


- If the symbol occurs more (with higher probability), the representing interval is larger

Arithmetic Coding

- Let $X = \{m_1, m_2, \ldots\}$ be a symbol set. The probability model for it is $P(X = m_i) = P(i)$
- The CDF of X is $F_X(i) = \sum_{j=1}^{i} P(j)$

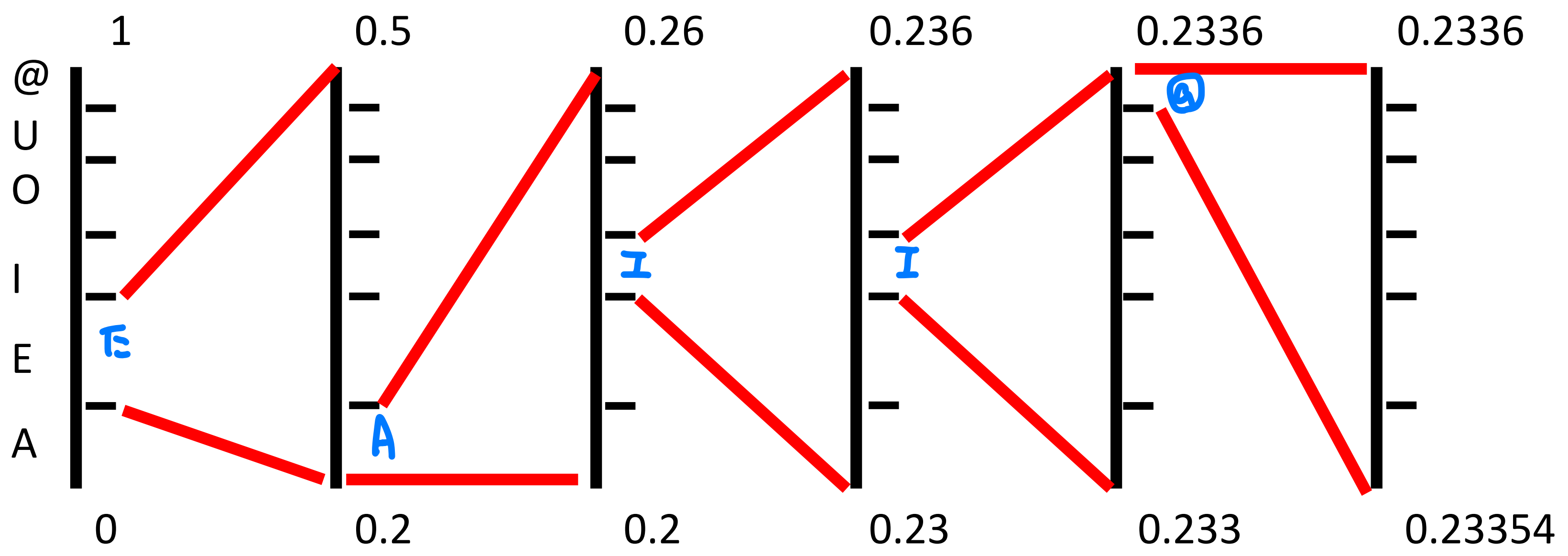
sym	A	B	C
prob.	0.2	0.3	0.5
F_x	0.2	0.5	1



tag for m_i can be chosen from any value that falls into $[F_X(i-1), F_X(i))$

Example

Encode “EAII@”:



X	Probability	Range
A	0.2	[0, 0.2)
E	0.3	[0.2, 0.5)
I	0.1	[0.5, 0.6)
O	0.2	[0.6, 0.8)
U	0.1	[0.8, 0.9)
@	0.1	[0.9, 1)

Arithmetic Coding – Example (cnt.)

- The final output interval is [0.23354,0.2336)
- Any number in the interval can represent the message "EAII@" as long as the decoder knows when to stop (here, "@" is the stopping symbol)
- Generating a tag for the message
  - We can choose the mid-point of the interval as the tag with the interval value **T** as

$$\boldsymbol{T_X(m_i) = F_X(i-1) + \frac{1}{2}P(i)}$$

$F_X(i)$

$p(i)$

$F_X(i-1)$

  - Then a binary code for **T** is the binary representation of the value truncated to fit the length of $\left\lceil \log\left(\frac{1}{P(x)}\right) \right\rceil + 1$ bits, where $P(x)$ is the probability of this message.

$$\left\lceil \log_2\left(\frac{1}{p(x)}\right) \right\rceil$$

Arithmetic Coding – Example (cnt.)

For example, $\mathcal{A} = \{ a_1, a_2, a_3, a_4 \}$ with probabilities $\{ 0.5, 0.25, 0.125, 0.125 \}$, a binary code for each symbol is as follows:

$\lceil \log \frac{1}{0.25} \rceil + 1$

Symbol	F_X	$\overline{T}_X$	In Binary	$\lceil \log \frac{1}{P(x)} \rceil + 1$	Code
1	.500	.2500	.0100	2	01
2	.750	.6250	.1010	3	101
3	.875	.8125	.1101	4	1101
4	1.000	.9375	.1111	4	1111

Generating a Binary Code

- After obtaining the interval, we can choose the mid-point of it as the tag, denoted as $\overline{\boldsymbol{T}}_{\boldsymbol{X}}(\boldsymbol{x})$

Let the interval be $\boldsymbol{l}^{(n)}$ and $\boldsymbol{u}^{(n)}$, $\boldsymbol{l}^{(n)} \leq \boldsymbol{t} < \boldsymbol{u}^{(n)}$ ($\boldsymbol{n}$ is the length of the symbol)

$$\overline{\boldsymbol{T}}_{\boldsymbol{X}}(\boldsymbol{x}) = \frac{\left(\boldsymbol{l}^{(n)} + \boldsymbol{u}^{(n)}\right)}{\mathbf{2}} = \frac{\left(\boldsymbol{F}_{\boldsymbol{X}}(\boldsymbol{x}-\mathbf{1}) + \boldsymbol{F}_{\boldsymbol{X}}(\boldsymbol{x})\right)}{\mathbf{2}}$$

A binary code for **T** is the binary representation of the value truncated to fit the length of $l(x) = \left\lceil \log(\frac{1}{P(x)}) \right\rceil + 1$ bits, where $P(x)$ is the probability of this message.

To show the code is uniquely decodable, we first show it is unique, meaning

$\lfloor \overline{\boldsymbol{T}}_{\boldsymbol{X}}(\boldsymbol{x}) \rfloor_{\boldsymbol{l}(\boldsymbol{x})}$ is unique if it is still falls into the interval $[F_X(i-1), F_X(i))$

Uniqueness of $\lfloor \overline{T}_X(x) \rfloor_{l(x)}$

To show $\lfloor \overline{T}_X(x) \rfloor_{l(x)}$ is unique, we only need to prove $F_X(x-1) \le \lfloor \overline{T}_X(x) \rfloor_{l(x)} < F_X(x)$

Since $\lfloor \overline{T}_X(x) \rfloor_{l(x)}$ is a truncated representation of $\overline{T}_X(x)$, we know

$$0 < \overline{T}_X(x) - \lfloor \overline{T}_X(x) \rfloor_{l(x)} \le \frac{1}{2^{l(x)}}$$

1. $\lfloor \overline{T}_X(x) \rfloor_{l(x)} < \overline{T}_X(x) = \frac{\left(F_X(x-1) + F_X(x)\right)}{2} < F_X(x)$
2. To show $\lfloor \overline{T}_X(x) \rfloor_{l(x)} \ge F_X(x-1)$

Uniqueness of $\lfloor \boldsymbol{\bar{T}_X(x)} \rfloor_{\boldsymbol{l(x)}}$

1. $\lfloor \boldsymbol{\bar{T}_X(x)} \rfloor_{\boldsymbol{l(x)}} < \boldsymbol{\bar{T}_X(x)} = \frac{\boldsymbol{(F_X(x-1)+F_X(x))}}{\boldsymbol{2}} < \boldsymbol{F_X(x)}$

2. To show $\lfloor \boldsymbol{\bar{T}_X(x)} \rfloor_{\boldsymbol{l(x)}} \geq \boldsymbol{F_X(x-1)}$

$$\frac{1}{2^{l(x)}} = \frac{1}{2^{\left\lceil \log_2\left(\frac{1}{P(x)}\right)\right\rceil+1}} < \frac{1}{2^{\log_2\left(\frac{1}{P(x)}\right)+1}} = \frac{1}{2\,\frac{1}{P(x)}} = \frac{P(x)}{2}$$

Since $\boldsymbol{\bar{T}_X(x) - F_X(x-1)} = \frac{\boldsymbol{(F_X(x-1)+F_X(x))}}{\boldsymbol{2}} - F_X(x-1) = \frac{\boldsymbol{(F_X(x-1)+F_X(x-1)+P(x))}}{\boldsymbol{2}} - F_X(x-1) = \frac{P(x)}{2}$

$$\boldsymbol{\bar{T}_X(x) - F_X(x-1)} > \frac{1}{2^{l(x)}}$$

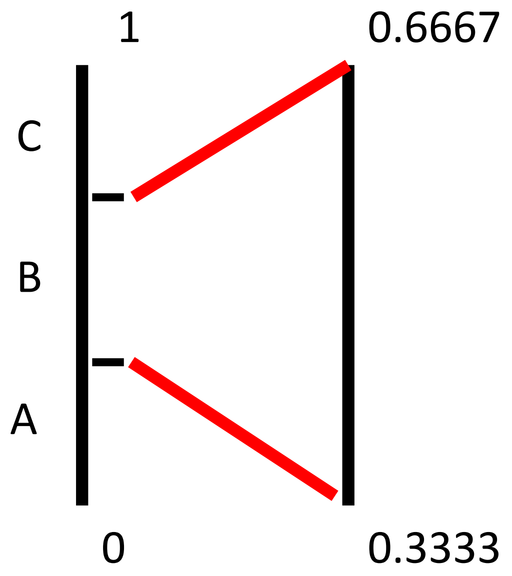
As $0 < \bar{T}_X(x) - \lfloor \bar{T}_X(x) \rfloor_{l(x)} \leq \frac{1}{2^{l(x)}} \rightarrow \ \bar{T}_X(x) - F_X(x-1) > \bar{T}_X(x) - \lfloor \bar{T}_X(x) \rfloor_{l(x)}$

$$\rightarrow \lfloor \bar{T}_X(x) \rfloor_{l(x)} > F_X(x-1)$$

Adaptivity to the data

- For example, when at time *t*

Encode "B":



Initial

X	Probability	Range
A	1/3	[0, 0.3333)
B	1/3	[0.3333, 0.6667)
C	1/3	[0.6667, 1)

Time *t*

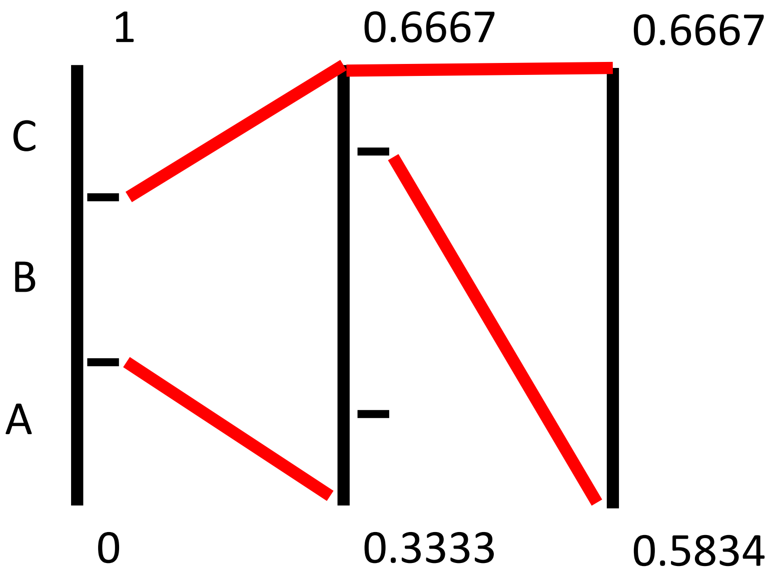
X	Probability	Range
A	1/4	[0.3333, 0.4167)
B	2/4	[0.4167, 0.5834)
C	1/4	[0.5834, 0.6667)

For next encoding

Adaptivity to the data

- For example, when at time *t+1*

Encode "C":



Time *t*

X	Probability	Range
A	1/4	[0.3333, 0.4167)
B	2/4	[0.4167, 0.5834)
C	1/4	[0.5834, 0.6667)

Time *t+1*

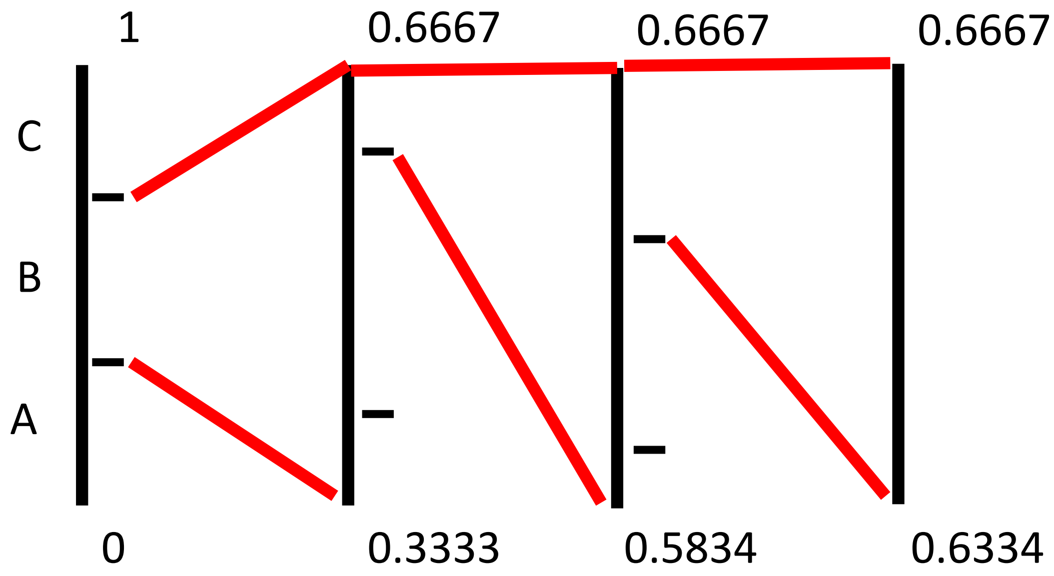
X	Probability	Range
A	1/5	[0.5834, 0.6001)
B	2/5	[0.6001, 0.6334)
C	2/5	[0.6334, 0.6667)

For next encoding

Adaptivity to the data

- For example, when at time *t+2*

Encode "C":



Time *t+1*

X	Probability	Range
A	1/5	[0.5834, 0.6001)
B	2/5	[0.6001, 0.6334)
C	2/5	[0.6334, 0.6667)

Time *t+2*

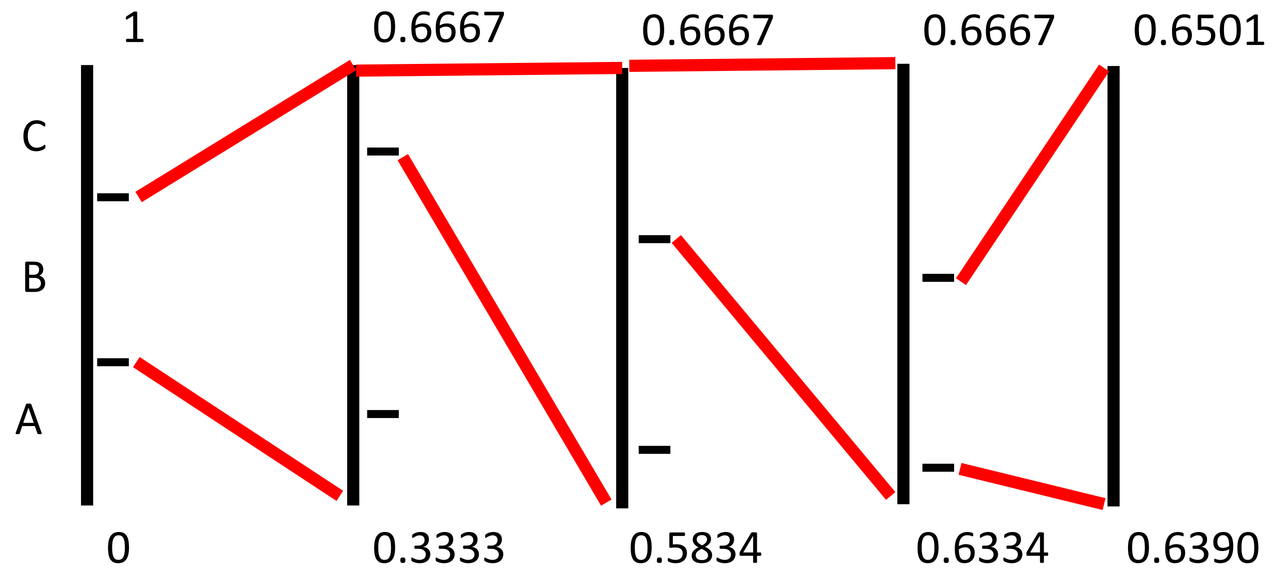
X	Probability	Range
A	1/6	[0.6334, 0.6390)
B	2/6	[0.6390, 0.6501)
C	3/6	[0.6501, 0.6667)

For next encoding

Adaptivity to the data

- For example, when at time *t+3*

Encode "B":



Time *t+2*

X	Probability	Range
A	1/6	[0.6334, 0.6390)
B	2/6	[0.6390, 0.6501)
C	3/6	[0.6501, 0.6667)

Time *t+3*

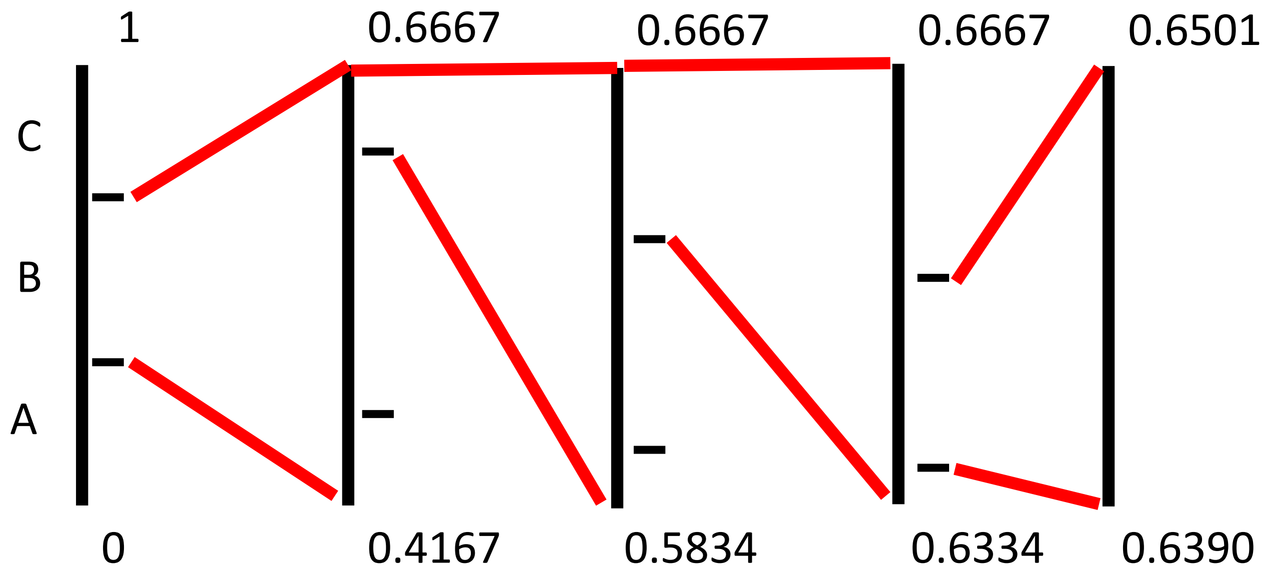
X	Probability
A	1/7
B	3/7
C	3/7

$$\frac{0.639 + 0.6501}{2} = 0.64455 \approx 0.64$$

Adaptivity to the data

- For example, when at time *t+3*

Encode “B”:



Time *t+2*

X	Probability	Range
A	1/6	[0.6334, 0.6390)
B	2/6	[0.6390, 0.6501)
C	3/6	[0.6501, 0.6667)

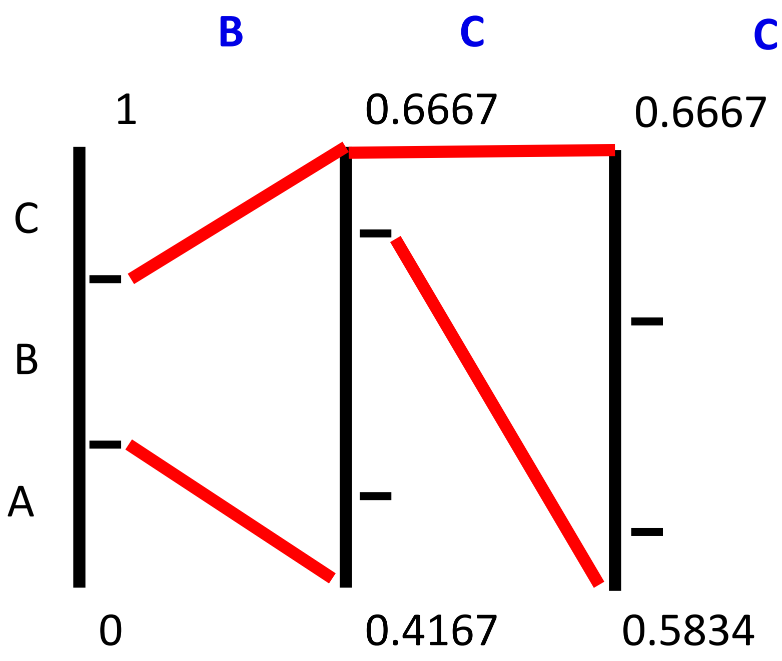
We can chose 0.64 to encode for “BCCB”

$$0.64_{10} \approx 0.1010001111_2$$
$$0.6396484375_{10}$$

Code: 1010001111

Decoding Process

- When receiving $0.1010001111_2 = 0.6396484375_{10}$



$\sim 0.63964_{10}$

Initial

X	Probability	Range
A	1/3	[0, 0.3333)
B	1/3	[0.3333, 0.6667)
C	1/3	[0.6667, 1)

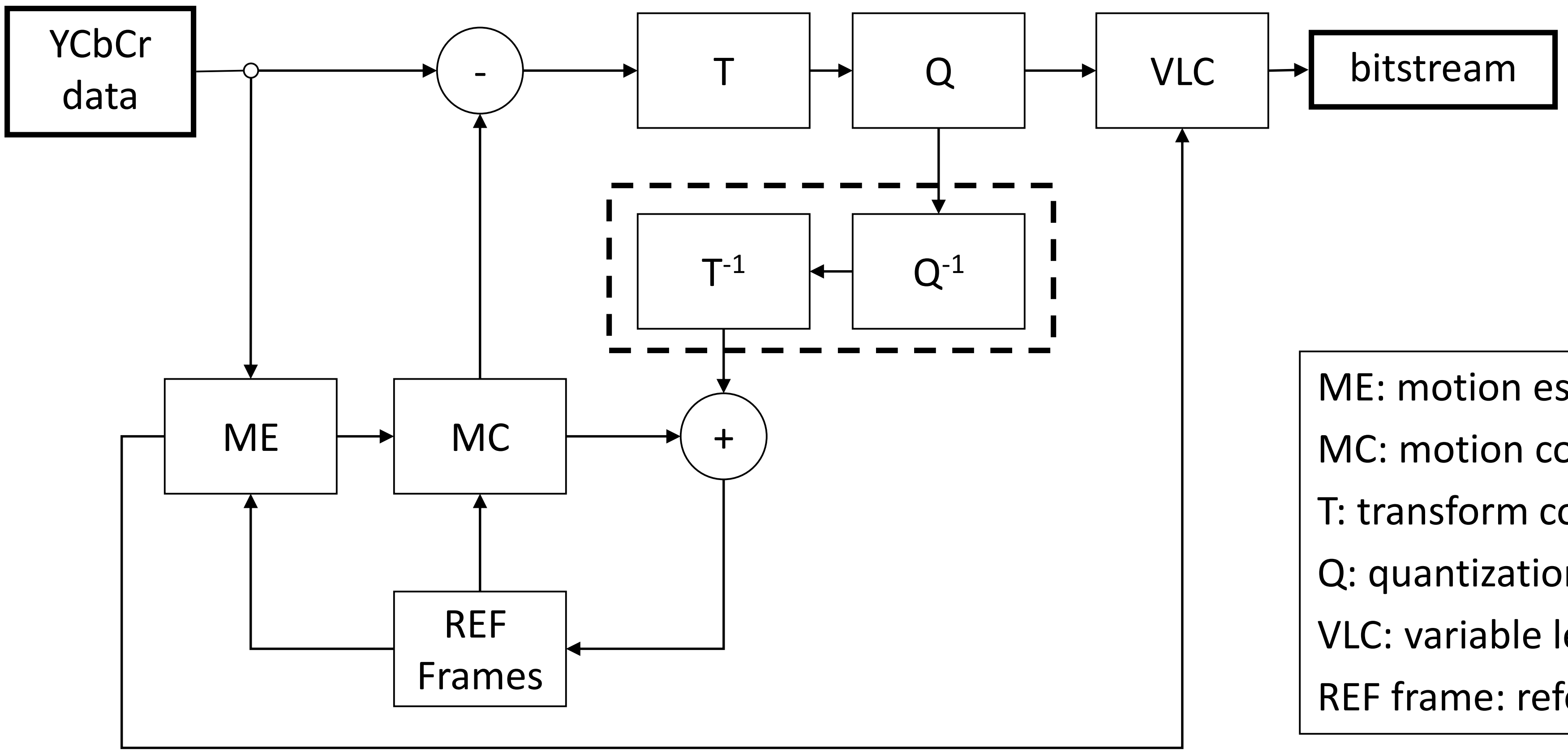
Once decoding 'B,' updating the table

X	Probability	Range
A	1/4	[0.3333, 0.4167)
B	2/4	[0.4167, 0.5834)
C	1/4	[0.5834, 0.6667)

Decoding 'C,' updating the table

X	Probability	Range
A	1/5	[0.5834, 0.6001)
B	2/5	[0.6001, 0.6334)
C	2/5	[0.6334, 0.6667)

Video Encoder Diagram



ME: motion estimation
MC: motion compensation
T: transform coding
Q: quantization
VLC: variable length coding
REF frame: reference frame

VLC in Video Coding

- Huffman coding is based on the statistics of the symbol occurrences
- Arithmetic coding can achieve better coding efficiency but more complex
- Unary and Golomb coding is usually used for encoding headers