Lab-01:

Theory:

Convolutional coding is an **error correction technique** used in digital communication systems to improve the reliability of data transmission.
Unlike block codes that encode data in fixed-size blocks, convolutional codes work on a **continuous stream of data bits**.

The encoder uses **shift registers** and **mod-2 adders (XOR gates)** to combine the current input bits with a few previous bits.
The resulting output sequence contains **redundant bits**, which help the receiver detect and correct errors using algorithms like the **Viterbi algorithm**.

Each convolutional code is defined by three parameters **(n, k, m)**:

- **k** → number of input bits at a time
- **n** → number of output bits generated
- **m** → number of memory elements

The **code rate (R)** is given by $R = k/n$, which represents the ratio of input bits to output bits.

Algorithm (Encoding Steps):

1. **Initialize** all shift registers to zero.
2. **Input** the data bits sequentially into the encoder.
3. For each new input bit:
   - Shift it into the register.
   - Generate output bits by performing **mod-2 addition** (XOR) of selected bits according to generator connections.
4. **Store** or **transmit** the output bits continuously.
5. After all bits are entered, **flush the encoder** by adding zeros to clear the registers.
6. The resulting sequence of output bits forms the **encoded data stream**.

Decoding Process:

1. **Receive** the encoded bit stream from the channel.
2. Use the **Viterbi algorithm** to find the most likely transmitted bit sequence.
3. **Compare** all possible paths through a **trellis diagram** and choose the path with the minimum error distance.
4. **Output** the estimated original data sequence.

Lab-02:

Theory:

The **Lempel–Ziv (LZ)** algorithm is a **universal lossless data compression method** developed by *Abraham Lempel* and *Jacob Ziv*. It is widely used in file compression formats such as **ZIP**, **GIF**, and **PNG**.

The Lempel–Ziv coding technique compresses data by identifying **repeated sequences** of symbols within a data stream and replacing them with **short references** (pointers) to previously seen data. This reduces redundancy and achieves efficient storage without losing information.

Algorithm (Encoding Steps):

1. **Initialize** the dictionary with all possible basic symbols (e.g., `0` and `1` for binary data).
2. **Start reading** the input data stream from left to right.
3. **Set** an empty string `S`.
4. **For each symbol** in the data:
   - Append the current symbol to `S`.
   - If `S` exists in the dictionary, continue reading the next symbol.
   - If `S` does not exist in the dictionary:
     - Add `S` to the dictionary as a new entry.
     - Output the index of the longest prefix of `S` found in the dictionary along with the new symbol.
     - Reset `S` to an empty string.
5. **Repeat** the process until all input symbols are processed.
6. **Output** the encoded stream as a sequence of index–symbol pairs or binary code blocks.

Decoding Algorithm:

1. **Initialize** the same dictionary used in encoding.
2. **Read** each index–symbol pair from the encoded stream.
3. **Look up** the dictionary entry corresponding to the index.
4. **Append** the symbol to that entry to form a new sequence.
5. **Output** the sequence and **add** it to the dictionary.
6. **Continue** until the entire encoded stream is decoded.

Labb-03:

Theory:

The **Hamming Code** is an **error-detecting and error-correcting code** introduced by *Richard W. Hamming* in 1950. It is widely used in data transmission systems to ensure reliable communication over noisy channels.

Hamming Code adds **redundant bits (parity bits)** to a data word to form a code word that can **detect and correct single-bit errors** and **detect two-bit errors**.

Basic Concept:
For a data word containing `k` data bits and `r` redundant bits, the total code length is:

$$n = k + r$$

The number of parity bits `r` is determined by the condition:

$$2^r \geq k + r + 1$$

Each parity bit covers certain positions in the code word, checking even or odd parity depending on the scheme used.

Algorithm (Encoding Process):

1. **Input:** The binary data word (e.g., `1011`).
2. **Determine** the number of parity bits `r` such that $2^r \geq k + r + 1$.
3. **Insert parity bits** into positions that are powers of 2 (1, 2, 4, 8, ...).
4. **Compute each parity bit** by checking specific positions according to binary representation rules:
   - Parity bit P_1 covers all positions with LSB = 1
   - Parity bit P_2 covers all positions with the second bit = 1
   - Parity bit P_4 covers all positions with the third bit = 1
   - and so on.
5. **Generate the final code word** by placing both data and parity bits together.

Decoding and Error Detection:

1. **Receive** the transmitted code word.
2. **Recalculate** parity bits using the same coverage pattern.
3. **Compare** calculated parity bits with the received ones:
   - If all parities match → **No error**.
   - If a mismatch occurs → **Form a binary error position number** using the mismatched parities.
4. **Correct** the bit at the error position (if any) by flipping its value.

Lab-04:

Theory:

The channel capacity C for a Binary Symmetric Channel (BSC), we need the channel transition probability matrix, which typically provides the probabilities of bit flips (errors) in a communication system.

For a Binary Symmetric Channel, the transition probability matrix is often represented as:

$$P(x) = \begin{pmatrix} 1-p & p \\ p & 1-p \end{pmatrix}$$

Here:

- p is the probability of a bit flip (i.e., the probability of an error).
- $1-p$ is the probability that the transmitted bit is received correctly.

The channel capacity C for a Binary Symmetric Channel is given by the formula:

$$C = 1 - H(p)$$

Where H(p) is the binary entropy function, defined as:

$$H(p) = -p \log_2 p - (1-p) \log_2 (1-p)$$

To find the channel capacity, follow these steps:

1. Identify the noise probability p from the noise matrix.
2. Calculate the binary entropy function $H(p)$.
3. Substitute H(p)H(p)H(p) into the channel capacity formula $C = 1 - H(p)$

Lab-05:

Theory:

Huffman Coding, developed by *David A. Huffman* in 1952, is a **lossless variable-length coding technique** used in data compression. It assigns **shorter codes** to more frequent symbols and **longer codes** to less frequent ones, minimizing the average code length.

The Huffman code is considered **optimal** when the average codeword length approaches the **entropy** of the source, meaning no other prefix code can achieve better compression efficiency.

Algorithm (Huffman Coding Steps):

1. **List all source symbols** with their corresponding probabilities or frequencies.
2. **Arrange** the symbols in ascending order of their probabilities.
3. **Combine** the two least probable symbols into a new node whose probability equals their sum.
4. **Repeat** step 3 until only one node (the root) remains, forming a binary tree.
5. **Assign binary codes**:
   - `0` for one branch
   - `1` for the other branch
6. **Read codes** from root to leaves to obtain the Huffman code for each symbol.
7. **Calculate average code length (Lavg):**

$$L_{avg} = \sum_{i=1}^{n} P_i \times L_i$$

   where P_i = probability of symbol *i*, and L_i = length of its code.

8. **Compute entropy (H):**

$$H = -\sum_{i=1}^{n} P_i \log_2 P_i$$

where P_i = probability of symbol *i*, and L_i = length of its code.

8. **Compute entropy (H):**

$$H = -\sum_{i=1}^{n} P_i \log_2 P_i$$

9. **Check optimality condition:**

$$H \leq L_{avg} < H + 1$$

If this holds true, the Huffman code is optimal.

Lab-06:

Definition: Entropy

The **entropy** $H(X)$ of a discrete random variable X is defined as:

$$H(X) = -\sum_{x \in X} P(x) \log_2 P(x)$$

It represents the **average information** or **uncertainty** associated with the random variable X.

Definition: Conditional Entropy

Given two random variables $(X, Y) \sim P(x, y)$, the **conditional entropy** $H(Y|X)$ is defined as:

$$H(Y|X) = \sum_{x \in X} P(x) H(Y|X = x)$$

$$= -\sum_{x \in X} \sum_{y \in Y} P(x, y) \log_2 P(y|x)$$

$$= -E[\log_2 P(Y|X)]$$

It measures the **average uncertainty in Y** when X is known.

Definition: Joint Entropy

The **joint entropy** of two random variables X and Y is given by:

$$H(X, Y) = -\sum_{x \in X} \sum_{y \in Y} P(x, y) \log_2 P(x, y)$$

It represents the total uncertainty of both variables considered together.

$$P(X) = \left(\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8}\right)$$

$$P(Y) = \left(\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}\right)$$

Calculations:

$$H(X) = \frac{7}{4} \text{ bits}, \quad H(Y) = 2 \text{ bits}.$$

Now,

$$H(X|Y) = \sum_{i=1}^{4} P(Y=i)\, H(X|Y=i)$$

$$= \frac{1}{4} H\left(\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8}\right) + \frac{1}{4} H\left(\frac{1}{4}, \frac{1}{2}, \frac{1}{8}, \frac{1}{8}\right) + \frac{1}{4} H\left(\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}\right) + \frac{1}{4} H(1,0,0,0)$$

$$= \frac{1}{4} \times \frac{7}{4} + \frac{1}{4} \times \frac{7}{4} + \frac{1}{4} \times 2 + \frac{1}{4} \times 0$$

$$H(X|Y) = \frac{11}{8} \text{ bits}.$$

Similarly,

$$H(Y|X) = \frac{13}{8} \text{ bits}, \quad H(X,Y) = \frac{27}{8} \text{ bits}.$$

Lab-07:

Theory:

The **entropy rate** of a stochastic process $\{X_i\}$ measures the **average uncertainty per symbol** in a long sequence of random variables. It is defined as:

$$H(X) = \lim_{n\to\infty} \frac{1}{n} H(X_1, X_2, ..., X_n)$$

when the limit exists.

Examples:

1. **Typewriter Model:**
   If a typewriter has m equally likely letters,

$$H(X_1, X_2, ..., X_n) = n \log_2 m$$

and

$$H(X) = \log_2 m \text{ bits per symbol.}$$

2. **IID Random Variables:**
   If $X_1, X_2, ...$ are independent and identically distributed (i.i.d.), then

$$H(X) = H(X_1)$$

Entropy Rate of a Random Walk on a Weighted Graph:
Consider a **graph with m nodes** labeled $\{1, 2, ..., m\}$ and **edge weights** $W_{ij} \geq 0$.
A random walk moves from node i to node j with probability proportional to the edge weight:

$$P_{ij} = \frac{W_{ij}}{\sum_k W_{ik}}$$

Let the total weight of edges from node i be:

$$W_i = \sum_j W_{ij}$$

and the total weight of all edges in the graph be:

$$W = \sum_i W_i/2$$

The **stationary distribution** is then:

$$\mu_i = \frac{W_i}{2W}$$

The **entropy rate** of the random walk is:

$$H(X) = H(X_2|X_1) = -\sum_i \mu_i \sum_j P_{ij} \log_2 P_{ij}$$

which simplifies to:

$$H(X) = H\left(\frac{W_{ij}}{2W}\right) - H\left(\frac{W_i}{2W}\right)$$

Special Case (Equal Weights):
If all edges have **equal weights**, and each node i has E_i edges while the total number of edges is E, then:

$$H(X) = \log_2 E - H\left(\frac{E_1}{2E}, \frac{E_2}{2E}, ..., \frac{E_m}{2E}\right)$$

This shows that the entropy rate depends only on the **stationary distribution** and the **total number of edges**.