

Practical 6

AIM:

Write a program to implement error detection and correction using HAMMING code concept. Make a test run to input data stream and verify error correction feature.

ALGORITHM:

1. Binary Conversion

- **Input:** Text message (txt).
- **Process:** Convert each character in the text to an 8-bit binary string.
- **Output:** Binary string representation of the message.

2. Calculate Number of Redundant Bits

- **Input:** Length of binary message (m).
- **Process:** Initialize $r = 0$, then iterate, incrementing r until $2^r \geq m + r + 1$.
- **Output:** Minimum number of redundant bits required (r).

3. Position Redundant Bits

- **Input:** Binary data and redundant bits count (r).
- **Process:**
 1. Initialize counters for data (k) and redundant bit positions (j).
 2. For each position i from 1 to $m + r$:
 - If i is a power of 2, place a '0' (initial redundant bit) and record the position in r_pos .
 - Else, add the next bit from bin_data .
- **Output:** Modified data with placeholder '0's for redundant bits and positions of redundant bits.

4. Calculate Parity Bits

- **Input:** Data with placeholder redundant bits and number of redundant bits (r).

- **Process:**
 1. For each redundant bit position (powers of 2):
 - Calculate the parity by XORing bits at positions where the bit position is set (i.e., $j \& pos$ is true).
 - Assign the calculated parity value to the redundant bit position in arr.
- **Output:** Data string with calculated parity bits inserted at redundant positions.

5. Induce Error

- **Input:** Encoded binary data with parity bits and error position.
- **Process:** Flip the bit at the specified position to simulate a transmission error.
- **Output:** Corrupted data with an error at the specified position.

6. Detect and Fix Errors

- **Input:** Corrupted binary data with parity bits and number of redundant bits (r).
- **Process:**
 1. Initialize $res = 0$ to store the error position.
 2. For each redundant bit position:
 - Calculate parity and XOR bits at the specified positions.
 - If parity check fails, add the position to res .
 3. If $res \neq 0$, flip the bit at position res to correct the error.
- **Output:** Corrected binary data

7. Remove Redundant Bits

- **Input:** Corrected binary data and number of redundant bits (r).
- **Process:** Remove bits at positions that are powers of 2.
- **Output:** Original binary data

8. Binary to Text Conversion

- **Input:** Corrected binary data without redundant bits
- **Process:** Convert each 8-bit segment of binary back to a character.

- **Output:** Decoded text message.

OUTPUT:

```
Enter text to be encoded: chanddraprakash
Positions of redundant bits: [1, 2, 4, 8, 16, 32, 64]
Parity bit in position 1: 1
Parity bit in position 2: 0
Parity bit in position 4: 1
Parity bit in position 8: 0
Parity bit in position 16: 1
Parity bit in position 32: 0
Parity bit in position 64: 1
Sender output (binary with redundant bits): 100111000011011101000011000010101011100110010001100100011100100111000001011100000111001001100001011010110110000101110011011
01000
Enter the bit position to introduce error: 5
Introduced error at position: 5
Error detected at position: 5
Data before correction: 100101000011011101000011000010101011100110010001100100011100100111000001011100100110000010110101101000010111001101101000
Error corrected at position: 5
Data after correction: 1001110000110111010000110000101010111001100100011001000111001001110000010111001001100000101101010110000010111001101101000
Decoded text: chanddraprakash

...Program finished with exit code 0
Press ENTER to exit console.
```