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:

# Convert Text to Binary:

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

# Calculate Number of Redundant Bits:

* Input: m (length of binary data)
* Process: Calculate the minimum number of redundant bits r required such that 2r≥m+r+12^r \geq m + r + 12r≥m+r+1.
* Output: r, the number of redundant bits.

# Position Redundant Bits in Binary Data:

* Input: data (binary data without redundant bits) and r
* Process:
  + Insert 0 at positions 2i2^i2i (1, 2, 4, 8, ...) to reserve space for redundant bits.
  + Keep track of these positions in r\_pos.
* Output: Binary data arr with placeholders for redundant bits and list r\_pos of their positions.

# Calculate Parity Bits:

* Input: arr (binary data with redundant bit placeholders) and r
* Process:
  + For each position 2i2^i2i, calculate parity by XOR-ing all bits covered by this position in binary (positions for which the bitwise AND with 2i2^i2i is non-zero).
  + Update each redundant bit placeholder in arr with the calculated parity value.
* Output: Binary data arr with calculated redundant (parity) bits.

# Sender Output:

* Print the final binary data with redundant bits added.

# Induce Error (Optional):

* Input: Binary data arr and error position pos
* Process: Flip the bit at position pos.
* Output: Corrupted binary data.

# Detect and Fix Error:

* Input: Corrupted binary data data and r
* Process:
  + For each position 2i2^i2i, calculate parity as in step 4.
  + Sum up positions of incorrect parity bits to find the error position res.
  + If res is non-zero, flip the bit at this position to correct the error.
* Output: Corrected binary data and the error position.

# Remove Redundant Bits:

* Input: Corrected binary data and r
* Process: Remove bits at redundant positions 2i2^i2i.
* Output: Original binary data without redundant bits.

# Convert Binary to Text:

* Input: Original binary data without redundant bits.
* Process: Split binary data into 8-bit chunks, convert each chunk to its ASCII character, and concatenate.
* Output: Decoded text.

# Display Results:

* Display the encoded binary data, induced error, error detection, correction process, and decoded text.

Output;

