Reg No: 220701053

**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. o 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). o 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. o 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.

**Code:**

import numpy as np

def txt\_to\_bin(txt):

return ''.join(format(ord(c), '08b') for c in txt)

def bin\_to\_txt(bin\_str):

chars = [bin\_str[i:i+8] for i in range(0, len(bin\_str), 8)]

return ''.join([chr(int(c, 2)) for c in chars])

def calc\_r\_bits(m):

r = 0

while (2\*\*r < m + r + 1):

r += 1

return r

def pos\_r\_bits(data, r):

j, k = 0, 0

m = len(data)

res = ''

r\_pos = []

for i in range(1, m + r + 1):

if i == 2\*\*j:

res += '0'

r\_pos.append(i)

j += 1

else:

res += data[k]

k += 1

# Print all positions of redundant bits in one line

print(f"Positions of redundant bits: {' '.join(map(str, r\_pos))}")

return res, r\_pos

def calc\_p\_bits(arr, r):

n = len(arr)

arr = list(arr)

parity\_bits\_info = []

for i in range(r):

p = 0

pos = 2\*\*i

for j in range(1, n+1):

if j & pos:

p ^= int(arr[j-1])

arr[pos-1] = str(p)

parity\_bits\_info.append(f"Parity bit in position {pos}: {p}")

# Print all parity bit information in one line

print(' '.join(parity\_bits\_info))

return ''.join(arr)

def detect\_and\_fix(data, r):

n = len(data)

res = 0

for i in range(r):

p = 0

pos = 2\*\*i

for j in range(1, n+1):

if j & pos:

p ^= int(data[j-1])

if p != 0:

res += pos

if res != 0:

print(f"Error detected at position: {res}")

data = list(data)

if res <= n:

data[res - 1] = '0' if data[res - 1] == '1' else '1'

print(f"Error corrected at position: {res}")

else:

print("Error position out of range. No correction performed.")

fixed\_data = ''.join(data)

print(f"Binary data after error correction: {fixed\_data}")

return fixed\_data

else:

print("No error detected.")

return data

def remove\_r\_bits(data, r):

j = 0

orig\_data = ''

for i in range(1, len(data) + 1):

if i == 2\*\*j:

j += 1

else:

orig\_data += data[i-1]

return orig\_data

def induce\_err(data, pos):

if pos < 1 or pos > len(data):

print("Error position is out of range.")

return data

data = list(data)

data[pos - 1] = '0' if data[pos - 1] == '1' else '1'

print(f"Introduced error at position: {pos}")

print(f"Binary data after introducing error: {''.join(data)}")

return ''.join(data)

def sndr(txt):

bin\_data = txt\_to\_bin(txt)

m = len(bin\_data)

r = calc\_r\_bits(m)

arr, r\_pos = pos\_r\_bits(bin\_data, r)

arr = calc\_p\_bits(arr, r)

print(f"Sender output (binary with redundant bits): {arr}")

return arr

def rcvr(data):

r = calc\_r\_bits(len(data))

fixed\_data = detect\_and\_fix(data, r)

orig\_data = remove\_r\_bits(fixed\_data, r)

ascii\_out = bin\_to\_txt(orig\_data)

print(f"Decoded text: {ascii\_out}")

if \_\_name\_\_ == "\_\_main\_\_":

inp\_txt = input("Enter text to be encoded: ")

ch\_data = sndr(inp\_txt)

err\_pos = int(input('Enter the bit position to introduce error: '))

corrupt\_data = induce\_err(ch\_data, err\_pos)

rcvr(corrupt\_data)

Output;

