EXPNO:06

16/08/24

Pratical-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 code.

Error correction at Data Link Layer:-

Hamming code is a set of error-correction codes that can be used to detect and correct the errors that can occur when data is transmitted from the sender to reciever. It is a technique develope by R.W. Hamming for error correction.

Create sender program with below feautres:-

1. Input to sender file should be a text of any length. Program should convert the text to binary.
2. Applying hamming code concept on binary data and add redundant bits to it.
3. Save this output in a file called channel.

Create a receiver program with below feautres:-

1. Receiver program should read the input from channel file.
2. Apply hamming code on binary data to check for errors.
3. If there is error, display the position of error.
4. Else remove the redudant bits and convert the binary data to ascii and display the output.

Student observation:-

Code:-

```
import math
def char_to_bin(ch);
    return [int(bit) for bit in format(ord(ch), '08b')]
def calc_parity(h code, n, r);
    for i in range(r):
      P_pos = 2**i;
      Parity = 0
    for j in range(p_pos, n+1; 2 * P_pos);
      for k in range(j, min(j + p_pos, n+1));
          Parity ^= h code[k].
    h code [p_pos] = parity
def gen_hamming(data);
      m = len(data)
      r = 0
      n = m
while n + r + 1 > 2 * * r;
      r + = 1
n = m + r
h code = [0] * (n+1)
      j = 0
      k = 0
      for i range (1, n+1):
        if i = = 2 * * k:
          h code [i] = 0
          k + = 1
        else:
          hcode [i] = data [j]
          j + = 1
      calc_parity(hcode, n, r)
      return hcode, n
```

```
def detect_and_correct(hcode, n, r):
    err_pos = 0
    for i in range(r):
        p_pos = 2**i
        parity = 0
        for j in range(p_pos, n+1, 2 * p_pos):
            for k in range(j, min(j+ p_pos, n+1)):
                parity ^= hcode[k]
        if parity != 0:
            err_pos += p_pos
    return err_pos

def bin_to char(bin_data):
    chars[]
    for i in range(0, len(bin_data), 8):
        ch = 0
        for j in range(8):
            ch |= bin_data[i+j] << (7-j)
            chars.append(chr(ch))
    return ''.join(chars)

def main():
    input_str = input("Enter input string:")
    bin_data[]
    for ch in input_str:
        bin_data.extend(char_to_bin(ch))
    hcode, n = gen_hamming(bin_data)

    print("Generated Hamming code:", ''.join(map(str,
                hcode[1:])))

    err_pos = int(input("enter position to error)).

    if any(err_pos ==2 **i for i in range(int
                (math.log2(n)) +1)):

    print(f'error cannot implement at redudant
                parity bit")
```

```
elif 0< err_pos <=n:
    hcode [err_pos] = 1 - hcode [err_pos]
    print ("Hamming code with error :", ' '.join
                    (map (str, hcode [1:])))

detected_err_pos = detect_and_correct (hcode, n,
                    int (math.log2 (n+1))).

if detected_err_pos = =0:
    print ("No error detected")

else:
    print (f"error detected at position: {err_pos}")
    hcode [detected_err_pos] = 1 - hcode [detected_err_pos]
    print ("corrected Hamming code:", ' '.join(map(str,
                    hcode[1:]))).

    print (f"corrected bit position {detected_err_pos}
                {hcode [detected_err_pos]}").

    corrected_data = [ ]
    j = 0
    k = 0
    for i in range (1, n+1):
        if i! = 2 ** k:
            corrected_data.append (hcode [i])
        else:
            k+ = 1
    corrected_str = bin_to_char (corrected_data)
    print ("corrected string:", corrected_str)

    if _name_ = = "_main_":
    main().
```

Output:

```
* Enter the input string: good
Generated Hamming code: 0100110 10111 0111
01111 0110 11110 10100 100.

Enter position to stimulate error : 5

Hamming code with error : 01000 1010 111 0111
01111 0110 11110 10 10 100 100

Error detected at position : 5

corrected Hamming code : 0100110 10 1110 111
01111 011 01111 0 1 0 100 100.

corrected bit at position 5 : 1

corrected string : good.
```

```
* Enter the input string: good
  Generated Hamming code: 0100110 10111 0111
  01111 0110 11110 1 0 100 100

  Enter position to stimulate error: 4
  Error cannot be implemented in redudant
  bit at position 4.
   no error detected.
   corrected string: good.
```

8 16/8/24

~~Output:~~ Result:-

Thus program is executed and output is verified successfully.