11.9.25 6. Error detection and correcting hamming code

Aim

write a program to implements error detection and correction using hamming code Concept. make a test run to input data stream and verify error and correction feature

Error correction at Data link Layer

Hamming code is set of error correcting codes that can be used to detect and correct the errors that can occur when the data is transmitted from the sender to the receiver

Procedure (Sender program)

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

Procedure (Receiver program)

1. Receiver program should read the input from channel file.
2. Apply hamming code on the library data to check for errors.
3. If there is an error display the position of the error.

4 Else remove the redundant bits and convert the library data to ascii and display the output

Program

Sender Pg

```
import math.
def str-to bits (s:str) -> list
    bits = [ ]
    for char s:
        b = format (ord (ch) '08b')
    bits extend ( int (x) for x in b ) to bits.
def add - parity - bits ( data - bits - list) -> list :
        m = len ( data - bits )
        r = 0
    while (2**r) < (m + r + 1) ;
        r + = 1
        total - len = m + r
            res = [ ]   j = 0
for, i in range ( 1, total - len + 1 )
        if (i == (j - 1)) == 0 ==
            res. append (0)
        else
            res - append (data - bits [ j ] )
                j + = 1
    for i in range (r);
            pos = 2 ** j
            count = 0
```

```
if j + pos and res[j-1] == 1:
    count + = 1
    res[pos-1] = count % 2.
    return res
def bits_to_str(bits: list) -> str:
    return "".join(str(b) for b in bits)
def main():
    S = input("Enter string to send: ")
    data_bits = str_to_bits(S)
    encoded = add_parity_bits(data_bits)
    bit_string = bits_to_str(encoded)
    with open('Code.txt', "w") as f:
    with open("original.txt", "w") as f:
        f.write(bit_string)
print("Encoded bit string written to Code.txt and original.txt")
print("Encoded (length {3}) = {3}".format
    (len(encoded), bit_string))
if __name__ == "__main__":
    main()
```

Receiver.py

```
import math
def read_bits_from_file(path="code.txt") ->
                                list):
    s = open(path).read().strip()
    Space - seperated
    s = "".join(s.split())
    return [int(ch) for ch in s]
def detect_and_correct(bits: list) ->
    (list, int);
        n = len(bits)
    r = 0
    while 2**r
        r <= n;
    r+1 = 1
    error_pos = 0
for i in range(r);
    pos 2**
    count = 0
for j in range(n);  pos = 2***;
    count = 0
for j in range(1, n+1);
if j & pos and bits[j-1] == 1
    count += 1
if count % 2 == 0
    error_pos += pos
```

```
if error_pos + = pos
    if error_pos ] = 0 =
    bits [error - pos - 1] ^ = 1
    return bits, error_pos
def extract_data_bits (bits ; list ) -> list
    n = len (bits)
    data = []
    for i in range(1, n+1];
        if (is (j-1)) ! = 0 ;
        data - append (bits (j-1)
    return data
def bits_to_string (data_bits ; list) -> str
    S = []
    extra = 8 - len (data_bits) % 8
if extra ;
    data_bits + = [0] * extra
for i in range (0, len (data_bits), 8 ) ;
    byte = (data_bits) (i : i + 8)
    val = 0
for b in byte
    val = (val << 1 ) | b
    st = (chr (val) return S
def main ( ) ;
    bits = read_bits_from_file ("code.txt")
print ("Received bit = {}", "join (str (b) for
    b bits ) )
    corrected, error_pos =
    detect_and_correct (bits)
    if error_pos = = 0.
```

```
print ("No single-bit error detected.")
else:
    print ("single-bit error detected and
corrected at position (1-based):", errorpos)
print ("corrected code:", ''.join (str(b) for b in
    corrected))
    data = extract data-bits (corrected)
  message = bits-to-string (data)
print ("Recovered message (may include
      padding):")
      print (message)
if __name__ == "__main__":
      main()
```

Sender output

```
Enter string to send - hi
Encoded bit string written to code.txt and
original.txt
Encoded (length 21) = 101001001001011000100
```

Receiver output

```
Received bits - 101001001001011000100
Single-bit error detected and corrected at
position (1-based) = 9
corrected cod = 101001001001011000100
Recovered message (may include padding) - hi
```

Result:

Thus the sender and receiver program for error detection and correction using Hamming code has been executed successfully.