INDEX

NAME : A.Kavin STD : III nd year SEC. : CSE-B ROLL NO. 220701122

Completed

Exp. NO : 06 Practical-6

Date : 16/08/2024

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 fautures

Error correction at Data link layer

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

Create sender program with below fautures

1. Input to sender file should be 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 channel.

Create ~~an~~ a receiver program with below feaut-ures

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 an error, display the position of the error

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

Student observation

code

```
import math
def char_to_binary(ch):
    binary=[]
    for i in range(7,-1,-1):
        binary.append((ord(ch)>>i)&1)
    return binary
def calculate_parity_bits(hamming_code,n,r):
    for i in range(r):
        parity_pos =2**i
        parity=0
        for j in range(parity_pos,n+1,2*parity_pos):
            for k in range(j,j+parity_pos):
                if k<=n:
                    parity^=hamming_code[k]
        hamming_code[parity_pos]=parity
def generate_hamming_code(data_bits,m):
    r=0
    n=m
    while n+r+1>2**r:
        r+=1
    n=m+r
    hamming_code =[0]*(n+1)
    j=0
    k=0
    for i in range(1,n+1):
        if i==2**k:
            k+=1
        else:
            hamming code[i]=data_bits[j]
            j+=1
calculate_parity_bits(hamming_code,n,r)
    return hamming_code,n,r
```

```
def detect_and_correct_error(hamming_code,n,r)
  error_pos=0
  for i in range(r):
    parity_pos=2**i
    parity=0
    for j in range(parity_pos, n+1,2*parity_pos):
      for k in range(j, j+ parity_pos):
        if k<=n:
          Parity ^=hamming_code[k]
      if parity !=0:
        error_pos +=parity_pos
    return error_pos
def binary_to_char(binary):
  output =
  for i in range (0, len(binary),8):
    ch=0
    for j in range(8)
      ch |=(binary[i+j] <<(7-j))
    output +=chr(ch)
    return output

def main():
  input_string= input("Enter the input string")
  binary =[]
  for ch in input_string:
    binary.extend(char_to_binary(ch))
  data_bits=binary[:]
  hamming code, n,r=generate_hamming_code
                       (date_bits, len(data_bits))
  Print("Generated hamming code:", ' '.join
         (map (str, hamming_code[1:])))
```

```
error_pos = -1
while True:
  error_pos = int(input("Enter the position to
          stimulate error (0 for no error):"))
   if error_pos > 0 and any(error_pos == 2**k for
          k in range(r)):
     Print("Error cannot be introduced in a
            redundant (parity) position. Please
            choose another position.")
     elif error_pos < 0 or error_pos > n:
       Print("Invalid position. Please enter
              a position between 1 and", n)
     else:
       break

if error_pos > 0:
   hamming_code[error_pos] ^= 1
   Print("Hamming code with error:", ''.join
    (map(str, hamming_code[1:])))
   detected_error_pos = detect_and_error_error
                  (hamming_code, n, r)
   if detected_error_pos == 0:
      Print("No error detected.")
   else
     Print(f"Error detected at position:
            {detected_error_pos}")
     binary_error_pos = format(detected_
                     error_pos, 'b')
     Print(f"corrected bit at position
  {detected_error_pos} (binary: {binary_error_
   pos}): {hamming code[detected_error_pos]}")
   hamming_code[detected_error_pos] ^= 1
   Print("corrected hamming code:", ''.
    join(map(str, hamming code[1:])))
```

```
corrected_data_bits=[]
k=0
for i in range(1, n+1):
    if i != 2**k:
        corrected_data_bits.append(hamming
                                      _code[i])
    else:
        k+=1
  corrected_string= binary_to_char(correcte
                                     -d_data_bits)
  print("corrected string:", corrected_strin
if __name__=="__main__":
  main()
```

Output

```
Enter the input string : apple
Generated Hamming code: 11001100000010
1111000000111000001110110001100101
Enter the position to stimulate error: 3
Hamming code with error :11101100000
101111000000111000000111000000111000000
1110110001100101
Error detected at position: 3
corrected bit at position 3 (binary: 11): 1
corrected hamming code: 11001100000010
11110000001110000001110110001100101
corrected string: apple
```

Result

Thus the program is executed successfully and output is verified

16/8/24