

Date: 21.8.29

Practical-6

Aim:- write a program to implement error detection & correction using Hamming code concept. Make a test to run input data stream

Error correction at data link layer;

Hamming code is a set of error correction codes that can be used to detect & correct errors that can occur when the data is transmitted from the sender to the receiver.

Student Observation:

Write the code here:

Sender.py (file name)

```
import os (stab)
def text_to_binary (text):
    return " ".join (format (ord (char), '08b') for char in text)
def calculate_redundant_bits (m):
    r = 0
    while (2 ** r) < (m + r + 1):
        r += 1
    return r
def position_redundant_bits (data, r):
    m = len (data)
    res = ''
    for i in range (1, m + r + 1):
        if i == 2 ** j - 1:
            res += '0'
        j += 1
```

else:

res += data[-k]

else k+1<sup>th</sup> position of message is stored  
insert return res[:k+1] returns a modified  
list of first k+1<sup>th</sup> elements of list

def calculate\_parity\_bits(data, r):  
from typing import List

n = len(data) # total no. of nonzeros in

For i in range(r): # else positions

val = 0 # no. of 1's in first i bits

for j in range(1, n+1): # for each bit

if j & (2\*\*i-1) == (2\*\*i-1): # if i<sup>th</sup> bit is set

val = val ^ int(data[-j]) # add 1<sup>st</sup> bit of value

arr = arr[:n-(2\*\*i)] + str(val) # add to arr

+ arr[n-(2\*\*i):] # add remaining bits

return arr

(def apply\_hamming\_code(data): # to append

m = len(data) # no. of total bits

r = m // 4 # calculated redundant bits (m)

arranged\_data = [data[i] for i in range(m)] # arranged data

hamming\_code = calculate\_parity\_bits(arranged\_data, r)

i = 0

: return hamming code

def save\_to\_channel(hamming\_code): # write

with open('channel', 'w') as file:

file.write(hamming\_code)

(absolutely)

if name == "main": # main

text = input("Enter the text: ")

binary\_data = text\_to\_binary(text)

hamming\_code = apply\_hamming\_code(binary\_data)

## save-to-channel (hamming-code)

# receiver.py

```
def read_from_channel():
    with open('channel', 'r') as file:
        return file.read()
```

```
def calculate_redundant_bit:
```

- 820  
while  $2^{r+1} < m+r+1$ :

$r+1$

return  $r$

```
def detect_error(data_nr):
```

$n = \text{len}(data)$

$res = 0$

for  $i$  in range( $nr$ ):

$val = 0$

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

if  $j \in (2^{r+1}, 2^{r+2})$ :

$val = val + \text{int}(data[-j])$

$val = val * 10^{(2^{r+1}-j)}$

$res = res + val$

return int(str(res), 2)

```
def correct_error(data, pos):
```

if  $pos > r$ :

~~data[0:n-r] = data[0:n-r] + str[1-ind]~~

~~data[n-r+1:n] = data[n-r+1:n] + str[1-ind]~~

~~return data~~

```
def remove_redundant_bit(data, nr):
```

$n = \text{len}(data)$

$j = 0$

$res = b$

for  $i$  in range( $1, n+1$ )

if  $i \neq 2^r + j$ :

$res + 2^{r+1} \cdot data[i]$

```

else: (don't print) len(s) <= 200
    f += 1
    return res[i:i+1]
def binary_to_text(binary_data):
    text = ''
    for i in range(0, len(binary_data), 8):
        byte = binary_data[i:i+8]
        text += chr(int(byte, 2))
    return text

```

Output:-

for compiling python sender.py

Enter the text : data  
 Data has been encoded and saved to  
 Channel file m1.hf  
 $\Rightarrow$  python receiver.py  
 Error detected at pos 8  
 Error is corrected.

Result:

The program was successfully  
 executed and o/p is verified

✓ working