16/8/24

PRACTICAL-6

EXPERIMENT ON ERROR DETECTION AND CORRECTION USING HAMMING CODE

Error detection at Data Link Layer:

Hamming code is a set of error-correctio 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. It is a technique developed by R.W. Hamming for error correction.

Create sender program with below features:

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 channel.

Create a receiver program with below features.

1) Receiver program should read the input from channel file.

2) Apply hamming code on the 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.

Program:

```
import math
def calculate_Redundant_bits(k):
    for r in range(1,k):
        if(2**r >= r+k+1):
            return r

def positioning_redundant_bit(data,r,k):
    result = []
    j=0
    p=k-1
    result.append('1')
    for i in range(1,k+r+1):
        if(2**j==i):
            j=j+1
            result.append('r')
        else:
            result.append(data[p])
            p=p-1
    return result[::-1], result

def building_ham_code(data,k,r):
    j=0
    c=0
    for t in range(1,k+r+1):
        if(t==2**j):
            l=2**j
            p=t
            c=0
            while(p<k+r+1):
                for h in range(p,p+1):
                    try:
```

```
            if (data[h]=='1' and p<k+r+1):
                c+=1
        except IndexError:
            break
    p=p+i+i
    if(c%2==0):
        data[i]='0'
    else:
        data[i]='1'
    j=j+1
if(flag==0):
    print("Built hamcode :"+"".join(data[::-1]))
return data

def checking_ham_code(data,k,r):
    j=0
    c=0
    for t in range(1,k+r+1):
        if(t==2**j):
            t=2**j
            p=t
            c=0
            while(p<k+r+1):
                for h in range(p,p+t):
                    try:
                        if(data[h]=='1' and p<k+r+1):
                            c+=1
                    except IndexError:
                        break
                p=p+t+t
            if(c%2!=0):
                pos=position_redbit(data,k,r)
```

```
            return pos
            break
        j = j+1

def position_redbit(data, k, r):
    j=0
    pos=0
    t=building_ham_code(data,k,r)
    for i in range(1, k+r+1):
        if(i==2**j):
            pos=(10**j)*int(data[i])+pos
            j=j+1
    print("Error position :"+str(int(str(pos),2)))
    return(int(str(pos),2))
c=input("Enter the string that you want to send: ")
data=''.join(format(ord(i),'08b') for i in c)
k=len(data)
r=calculate_Redundant_bits(k)
print("The string in Binary form: "+"".join(data))
data, result=positioning_redundant_bit(data,r,len(data))
print("The string after redundant bit is Inserted :
                                    "+"".join(data))

flag=0
data=building_ham_code(result, k, r)
flag=1
print("Transmission Time ....")
n=int(input("Enter the position to change during
                                    transmission : "))
if(math.sqrt(n)==math.floor(math.sqrt(n))):
    print("You can't change the redundant bit
                try some other position.")
else:
```

```
data[n]='0' if(data[n]=='1') else '1'
print("After error : "+"".join(data[::-1]))
h=" ".join(data[::-1])
pos=checking_ham_code(data,k,r)
h=list(h)
h[pos-1]='0' if(h[pos-1]=='1') else '1'
print("The corrected Error string : "+"".
                    join(h[::-1]))
```

Output :

Enter the string that you want to send : read

the string in binary form :
01110010011001 01011001 00

The string after redundant bit is inserted :
01110100110010101101010

Built hamcode : 01110010011000101011010
10100000

Transmission time.....

Enter the position you want to change : 3

20/8/24

After error : 01110010011000101011010100100
Error position : 3
corrected string : 01110010011000001011010100000

Result :

The code of building & checking the hamming code is successfully executed and the output is verified.