HAMMING CODE

apsara

Date

Experiment - 6

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 the data is transmitted from the sender to the receivers. 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 reciever 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 errors.

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

apsara

Date:

Code :-

```
def calc_r(l):
    for i in range(l):
        if (2**i >= l+i+1):
            return i

def pos_red_bits(b,r):
    j,k = 0,1
    bits = ""

    for i in range (1, len(b)+r+1):
        if (i == 2**j):
            bits += "0"
            j+=1
        else:
            bits += b[-1*k]
            k+=1
    return bits[::-1]
def calc_parity(arr, r):
    n = len(arr)
    for i in range(r):
        val=0
        for j in range (1, n+1):
            if (j & (2**i) == [2**i]):
                val = val ^ int[arr[-1*j]]
        arr = arr [:n-(2**i)] + str(val) + arr[n-(
    return arr                                  +1:]
def detect_err (arr, r):
    n = lenn(arr)
    res=0
    for i in range (r):
        val=0
        for j in range (1, n+1):
            if (j & (2**i) == (2**i)):
```

Page No.

```
            val = val ^ int ( arr [
        res = res + val * (10 ** i
    return int (str(res)
def flip (data, pos):
    if pos < 1 or pos > len
        print (" Invalid
        return data.
    data_list = list (data)
    data_list [pos-1] = '1'
    return "".join (data_list)
def bin_to_dec (b):
    return int (b,2)
S = input (" Enter a
bin_val = "".join ([bin (or
print ("Binary represen
l = len (bin_val)
r = calc_r (l)
print ("Number of redun
pos = pos_red_bits (bin_
enc_data = calc_parity
print (" Data with red
while True:
    err_pos = int (input (
    of the bit to flip (1-ba
    data)}) : "))
    if err_pos in [2 ** i for
        print (" Cannot fli
    position. please enter
        continue
    else:
        enc_data_err =
        print(f" Data with
        break
```

```
        val = val ^ int(arr[-1*j])
    res = res + val * (10**i)
    return int(str(res), 2)
def flip(data, pos):
    if pos < 1 or pos > len(data):
        print("Invalid position!")
        return data
    data_list = list(data)
    data_list[pos-1] = '1' if data_list[pos-1] == '0' else '0'
    return ''.join(data_list)
def bin_to_dec(b):
    return int(b,2)
s = input("Enter a string to encode: ")
bin_val = ''.join([bin(ord(c))[2:].zfill(8) for c in s])
print("Binary representation of '{s}': {bin_val}")
l = len(bin_val)
r = calc_r(l)
print("Number of redundant bits: {r}")
pos = pos_red_bits(bin_val, r)
enc_data = calc_parity(pos, r)
print("Data with redundant bits: {enc_data}")
while True:
    err_pos = int(input(f"Enter the position of the bit to flip (1-based index, 1 to {len(enc_data)}): "))
    if err_pos in [2**i for i in range(r)]:
        print("Cannot flip a redundant bit position. please enter a valid position")
        continue
    else:
        enc_data_err = flip(enc_data, err_pos)
        print(f"Data with error introduced: {enc_data_err}")
        break
```

```
err_detected = detect_err(enc_data,err)
if err_detected != 0:
    print("No error detected in the received data.")
else:
    err_pos_detected = err_detected
    err_pos_left = len(enc_data_err) - err_pos_
                   detected + 1
    bin_err_pos = bin(err_pos_left)[2:].zfill(4
    dec_err_pos = bin_to_dec(bin_err_pos)
    print(f"Error detected at position:
                {err_pos_left}")
    print(f"Binary error position: {bin_err_
                Decimal position: {dec_err_pos}")
    correct = input("Do you want to correct
the error? (yes/no):").strip().lower()
    if correct == 'yes':
        corrected_data = flip(enc_data_e
                              err_pos_left)
        print(f"Corrected data: {corrected_
                                  data}")
    else:
        print("Error was not corrected")
```

output :-

```
Enter the string : Hi
Binary : 0100100001101001
Hamming code : 01001001000011010 01
Flip a bit for error...
Flip bit (1-21) : 2
Redundant bit choose another Positio
Flip a bit for error...
Flip bit (1-21) : 3
Error!
```

Hamming code with error: 01101001100011001001
Error at: 3
Binary Pos: 00011
corrected code 01001001100001001001
Final output: Hi

Result:

Thus, the program for Hamming code for implementing error detection & correction is successfully executed & output is verified.