29/4/24

Hamming Code

Aim:

To write a hamming code to find & replace a single bit error

Code:

```
def string_to_binary (input_string):
    return ''.join(format(ord(c), '08b') for
                   c in input_string)

def binary_to_string (binary_data):
    chars = []
    for i in range (0, len (binary_data), 8):
        byte = binary_data [i:i+8]
        chars.append (chr (int (byte, 2)))
    return ''.join (chars)

def calc_parity_bits (data):
    n = len(data)
    r = 0
    while (2**r) < (n+r+1):
        r += 1
    return r

def insert_parity_bits (data, r):
    n = len(data)
    j = 0
```

```
k=0
n=m+r
hamming_code = []

for i in range (1, n+1):
    if i == 2**j:
        hamming_code.append(0)
        j+=1
    else:
        hamming_code.append(int
            (data[k]))
        k+=1
return hamming_code

def calc_pos_val (hamming_code, n):
    n= len(hamming_code)

    for i in range(r):
        parity_pos = 2**i
        parity_val = 0
        for j in range (1, n+1):
            if j & parity_pos and j != parity_pos
                parity_val ^= hamming_code[j-
        hamming_code[parity_pos-1] = parity_v

    return hamming_code
```

```
def detect_and_correct_error(hamming_code, n):
    n = len(hamming_code)
    error_position = 0

    for i in range(n):
        parity_pos = 2**i
        parity_val = 0
        for j in range(1, n+1):
            if j & parity_pos:
                parity_val ^= hamming_code[j-1]

    if parity_val != 0:
        error_position += parity_pos

    if error_position:
        print("Error detected at position:
                {error_position}")

        hamming_code[error_position-1] ^= 1

        print("Corrected Hamming code:
                {hamming_code}")

    else:
        print("No error detected")

    return hamming_code
```

```
def extract_data_from_hamming(hamming_code):
    j = 0
    data = []
    for i in range(1, len(hamming_code)+1):
        if i != 2**j:
            data.append(hamming_code[i-1])
        else:
            j += 1
    return ''.join(map(str, data))

def main():
    input_string = "Myself Kunal"
    binary_data = string_to_binary(input_string)
    print("Binary representation of {input_string} is {binary_data}")
    r = calculate_parity_bits(binary_data)
    hamming_code = insert_parity_bits(binary_data, r)
    hamming_code = calculate_parity_values(hamming_code)
```

corrected_string = binary_to_string(
"corrected_bin_data)
(corrected_string")

if __name__ == "__main__":
main().

Output:

Binary representation of 'Myself Kamal'
0100 11010111 00101111 01001001101100001001
1000100000 1000000 100 1011000010110100
00 10110110 0

Hamming code with parity bits: [0,1,0,0,1,1,1,0,
0,1,0,1,0,0,1,1,0,0,1,0,0,1,0,0,1,1,0,0,1,0,
0,1,0,1,0,1,1,1,0,0,1,0,1,1,1,0,0,0,0,0,1,0,0,0,0,1,1,
0,0,1,0,1,0,1,1,1,0,0,0,0,0,0,1,0,0,0,0,1,1,
1,1,0,0,0,0,1,1,0,1,1,0]

Enter the bit position (1-103) to introduce
error: 6

Hamming code with error: [1,1,0,1,1,0,0,0...
...0,1,1,0,0]

20/8/24

Error detected at position: 6

Final output after correcting: "Myself Kamal"

Result:
Thus it is successfully executed & output is verified.