op = int(input("Enter 1 for Hamming code generation\nEnter 2 for error detection\n"))

if op == 1:

m = list(map(int, input("Enter the data bits in binary:\n"))) r = 0

while (len(m) + r + 1) > (2 \*\* r): r += 1

print("Total number of data bits m =", len(m)) print("Total number of parity bits required r =", r)

print("Total number of bits in the encoded data =", len(m) + r) print("The redundant bits are placed in the position", [2 \*\* x

for x in range(r)])

m.reverse()

c, ch, j, hamming = 0, 0, 0, []

for i in range(0, (r + len(m))): p = (2 \*\* c)

if p == (i + 1): hamming.append(0) c = c + 1

else:

hamming.append(int(m[j])) j = j + 1

for parity in range(0, len(hamming)): ph = (2 \*\* ch)

if ph == (parity + 1): startIndex = ph - 1 i = startIndex

y = []

while i < len(hamming): block = hamming[i:i + ph] y.extend(block)

i += 2 \* ph

for z in range(1, len(y)):

hamming[startIndex] = hamming[startIndex] ^ y[z] ch += 1

hamming.reverse()

print('Hamming code generated would be:', end="") print(int(''.join(map(str, hamming))))

elif op == 2:

print('Enter the received Hamming code') d = input()

data = list(d) data.reverse()

c, ch, h, h\_copy = 0, 0, [], []

for k in range(0, len(data)): p = (2 \*\* c) h.append(int(data[k])) h\_copy.append(data[k]) if p == (k + 1):

c = c + 1 parity\_list = []

for parity in range(0, len(h)): ph = (2 \*\* ch)

if ph == (parity + 1): startIndex = ph - 1 i = startIndex

y = []

while i < len(h):

block = h[i:i + ph] y.extend(block)

i += 2 \* ph

for z in range(1, len(y)):

h[startIndex] = h[startIndex] ^ y[z] parity\_list.append(h[parity])

ch += 1 parity\_list.reverse()

error = sum(int(parity\_list) \* (2 \*\* i) for i, parity\_list in enumerate(parity\_list[::-1]))

if error == 0:

print('There is no error in the received Hamming code') elif error >= len(h\_copy):

print('Error cannot be detected') else:

print('Error is in', error, 'bit')

else:

if h\_copy[error - 1] == '0':

h\_copy[error - 1] = '1'

elif h\_copy[error - 1] == '1':

h\_copy[error - 1] = '0'

print('After correction, Hamming code is:') h\_copy.reverse() print(int(''.join(map(str, h\_copy))))

print('Option entered does not exist')

'''

python -u "C:/Users/Rishab/OneDrive/Desktop/CN Experiments/import hamm.py"

Enter 1 for Hamming code generation

Enter 2 for error detection

1

Enter the data bits in binary:

1101

Total number of data bits m = 4

Total number of parity bits required r = 3

Total number of bits in the encoded data = 7

The redundant bits are placed in the position [1, 2, 4]

Hamming code generated would be:1100110

python -u "C:/Users/Rishab/OneDrive/Desktop/CN Experiments/import hamm.py"

Enter 1 for Hamming code generation

Enter 2 for error detection

2

Enter the received Hamming code

1100110

There is no error in the received Hamming code

python -u "C:/Users/Rishab/OneDrive/Desktop/CN Experiments/import hamm.py"

Enter 1 for Hamming code generation

Enter 2 for error detection

2

Enter the received Hamming code

1100111

Error is in 1 bit

After correction, Hamming code is:

1100110

'''