def text\_to\_binary(text):

    binary\_data = ''

    for char in text:

        binary\_data += format(ord(char), '08b')  # Convert to 8-bit binary

    return binary\_data

def calculate\_redundant\_bits(m):

    for i in range(m):

        if (2\*\*i >= m + i + 1):

            return i

def generate\_hamming\_code(data):

    m = len(data)

    r = calculate\_redundant\_bits(m)

    arr = [0] \* (m + r)

    j = 0

    for i in range(1, m + r + 1):

        if i == 2\*\*j:  # Reserve positions for redundant bits

            j += 1

        else:

            arr[i - 1] = int(data[i - j - 1])  # Fill data bits

    for i in range(r):

        arr[(2\*\*i) - 1] = calculate\_parity(arr, i, m, r)  # Set parity bits

    return arr

def calculate\_parity(arr, i, m, r):

    val = 0

    for j in range(1, m + r + 1):

        if j & (2\*\*i) == (2\*\*i):  # Check parity bit positions

            val ^= arr[j - 1]

    return val

def sender():

    # Input data

    text = "Giri"

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

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

    with open("Channel.txt", "w") as f:

        f.write(''.join(map(str, hamming\_data)))

    print("Data sent and saved to Channel.txt:", hamming\_data)

def read\_channel():

    with open("Channel.txt", "r") as f:

        return f.read()

def detect\_error(arr, nr):

    n = len(arr)

    res = 0

    for i in range(nr):

        val = 0

        for j in range(1, n + 1):

            if j & (2\*\*i) == (2\*\*i):

                val ^= int(arr[j - 1])

        res += val \* (10\*\*i)

    return int(str(res), 2)

def correct\_error(arr, pos):

    if pos != 0:

        arr[pos - 1] = str(1 - int(arr[pos - 1]))  # Flip the bit at the error position

    return arr

def remove\_redundant\_bits(arr, r):

    n = len(arr)

    res = ''

    j = 0

    for i in range(1, n + 1):

        if i != (2\*\*j):  # Skip redundant bits

            res += arr[i - 1]

        else:

            j += 1

    return res

def binary\_to\_text(binary\_data):

    text = ""

    for i in range(0, len(binary\_data), 8):

        byte = binary\_data[i:i+8]  # Convert 8-bit binary to text

        text += chr(int(byte, 2))

    return text

def receiver():

    hamming\_data = read\_channel()

    nr = calculate\_redundant\_bits(len(hamming\_data))

    error\_pos = detect\_error(hamming\_data, nr)

    if error\_pos == 0:

        print("No error detected.")

        correct\_data = hamming\_data

    else:

        print(f"Error detected at position: {error\_pos}")

        correct\_data = correct\_error(list(hamming\_data), error\_pos)

    data\_without\_redundant\_bits = remove\_redundant\_bits(correct\_data, nr)

    ascii\_text = binary\_to\_text(data\_without\_redundant\_bits)

    print(f"Received text: {ascii\_text}")

# Example usage

sender()

receiver()

