src\hamming.py

```python
import numpy as np

G = np.array([
    [1,0,0,0,0,1,1],
    [0,1,0,0,1,0,1],
    [0,0,1,0,1,1,0],
    [0,0,0,1,1,1,1]
], dtype=int)  # data->code mapping (rows=data bit)

H = np.array([
    [0,1,1,1,1,0,0],
    [1,0,1,1,0,1,0],
    [1,1,0,1,0,0,1]
], dtype=int)  # parity-check matrix (3 x 7)

# syndrome table for Hamming(7,4)
SYNDROME_TO_ERROR_POS = {
    (0,0,0): None,
    (0,0,1): 6,
    (0,1,0): 5,
    (0,1,1): 2,
    (1,0,0): 4,
    (1,0,1): 0,
    (1,1,0): 1,
    (1,1,1): 3
}

def encode_nibble(nibble_bits):
    """Encode 4-bit list/array to 7-bit list"""
    d = np.array(nibble_bits, dtype=int).reshape(4)
    cw = (d.reshape(1,4).dot(G) % 2).reshape(7)
    return cw.tolist()

def decode_codeword(codeword_bits):
    """Decode 7-bit codeword -> corrected 4-bit data and flag if single-bit error corrected"""
    r = np.array(codeword_bits, dtype=int).reshape(7)
    syndrome = tuple((H.dot(r) % 2).tolist())
    pos = SYNDROME_TO_ERROR_POS.get(syndrome, None)
    corrected = False
    if pos is not None:
        # correct single-bit error in position pos (0..6)
        r[pos] ^= 1
        corrected = True
    # Recover data bits (systematic mapping: first 4 bits are data)
    data_bits = r[:4].tolist()
    return data_bits, corrected

def encode_bytes(data_bytes):
    """
    encode bytes object into bitarray (Hamming(7,4) per nibble)
    returns python list of ints 0/1
```

```
    """
    out = []
    for b in data_bytes:
        high = [(b >> 7) & 1, (b >> 6) & 1, (b >> 5) & 1, (b >> 4) & 1]
        low = [(b >> 3) & 1, (b >> 2) & 1, (b >> 1) & 1, b & 1]
        out.extend(encode_nibble(high))
        out.extend(encode_nibble(low))
    return out

def decode_bits_to_bytes(encoded_bits):
    """
    encoded_bits: list of ints length multiple of 7
    returns bytes and statistics (corrected_count)
    """
    corrected_count = 0
    out_bytes = []
    bits = encoded_bits
    for i in range(0, len(bits), 14):
        # two codewords => one byte
        if i+14 > len(bits):
            break
        cw1 = bits[i:i+7]
        cw2 = bits[i+7:i+14]
        d1, c1 = decode_codeword(cw1)
        d2, c2 = decode_codeword(cw2)
        corrected_count += (1 if c1 else 0) + (1 if c2 else 0)
        b = 0
        for bit in d1:
            b = (b << 1) | int(bit)
        for bit in d2:
            b = (b << 1) | int(bit)
        out_bytes.append(b)
    return bytes(out_bytes), corrected_count
```