Natalie Valett, nvalett
CSE13S
Prof. Darrell Long
April 19, 2021

Asgn5 Design: Hamming Codes

Summary:

This program is meant to provide an interface which will encode and decode Hamming Codes as a method of error correction in data. The Hamming Code being implemented will be a systematic code where the parity bits are placed after the data bits, and in format Hamming(8,4), or containing 4 data bits and 4 parity bits, for a total of 1 byte.

To Encrypt:

For each 4 data bits, the full encrypted byte should be composed accordingly:

Index	7	6	5	4	3	2	1	0
Hamming code	P_3	P_2	P_1	P_0	D_3	D_2	D_1	D_0

With P_0, P_1, P_2, P_3 corresponding to these values:

$$P_0 = D_0 \oplus D_1 \oplus D_3$$
$$P_1 = D_0 \oplus D_2 \oplus D_3$$
$$P_2 = D_1 \oplus D_2 \oplus D_3$$
$$P_3 = D_0 \oplus D_1 \oplus D_2 \oplus D_3 \oplus P_0 \oplus P_1 \oplus P_2$$

In order to encrypt the data into our Hamming Code, the data given will be represented via bit matrices, which are composed of bit vertices. When given a nibble of data m, we generate its hamming code c by performing matrix multiplication of **m * G**, where G is the generator matrix:

$$\boldsymbol{G} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}.$$

The result is encrypted bit vector c.

Psuedocode:

```
u8 ham_encode(BitMatrix *G, uint8_t msg):
        Msg_matrix = bm_from_data(msg)
        c = matrix_multiplication(G, msg_matrix)
```

```
return bm_to_data(c)
```

To Decrypt:

To decode a message, encrypted message c is multiplied by the *transpose* of the parity-checker matrix H:

$$\boldsymbol{H} = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

This multiplication results in a BitVector which represents the error syndrome of the message and tells us which bit needs to be flipped in order to correct the error. For example, if e = [0, 1, 1, 1], this tells us that the 0th bit is incorrect because e is equal to the 0th row in H^(transposed), and that bit must be flipped in order to correct the message error. If it returns 0 there's no errors and we can simply parse the message from vector c.

Psuedocode:

```
HAM_STATUS ham_decode(BitMatrix *Ht, uint8_t code, uint8_t *msg):
    // Load corrected msg into *msg, return HAM_STATUS of decrypt
    HAM_STATUS status;
    Int bit_to_flip;
    Uint8_t err_syndrome
    BitMatrix *c = bm_from_data(code)
    err_syndrome = matrix_multiplication(Ht, c)
    If err_syndrome == 0:
        status = HAM_OK
    Else:
        result = lookup(err_syndrome)
    If result.is_digit():
        flip_bit(result);
        Status = HAM_CORRECT
    Else: // if result isn't a digit, it's a ham status
        Status = result
```

Bit Handling:

```
BitVector {
    u32 length
    u8 *vector;
```

```
}
set_bit(n):
        byte = n / 8
        offset = n % 8
        shift a 0x01 to the index
        perform bitwise or with original code
*Alternative*:
set_bit(n):
        v->vector[n / 8] |= (1 << (n % 8))

u8 bm_to_data(bm):
        u8 msg;
        For i in bm:
                msg |= bm_get_bit(bm, i / 8, i % 8)
                msg = msg << 1

bm_multiply(mb A, bm B):
        // check that A->cols == B->rows
```

Abstract Data Types:

- BitVector: a 1D array of bits
- BitMatrix: an array of BitVectors (so a 2D array)

Command-line Options:

- -h: Prints out a help message describing the purpose of the program and the command-line op-tions it accepts, exiting the program afterwards.
- -i infile: Specify the input file path containing data to encode into Hamming codes. The default input should be set as stdin.
- -o outfile: Specify the output file path to write the encoded data (the Hamming codes) to. If not specified, the default output should be set as stdout.

ONLY FOR DECODE:

- -v: Prints statistics of the decoding process to stderr. The statistics to print are the total bytes processed, uncorrected errors, corrected errors, and the error rate. The error rate is defined as(uncorrected errors/total bytes processed), the ratio of uncorrected errors to total bytes processed.

Hamming Code:

RUNNER FILES:
encode.c:
encoder main():

```
    getopt() loop
    bm G = generator matrix
    while (msg = fgetc) != EOF:
        Byte = (MSB) 1100 0110 (LSB)
        msg1 = lower_nibble(msg) // low nibble of byte: 0110
        Msg2 = upper_nibble(msg) // high nibble of byte: 1100
        Code1 = ham_encode(G, msg1)
        code2 = ham_encode(G, msg2)
        fputc(code1, code2)
```

decode.c:
decoder main():

```
    getopt() loop
    bm Ht = BitMatrix
    While (code = getc) != EOF:
        get code1 // low nibble
        get code 2 // high nibble
        Msg1 = decode(Ht, code1)
        Msg2 = decode(Ht, code2)
        pack_byte(msg2, msg1)
        fputc(packed_nibbles)
```

PRE-LAB QUESTIONS:

1. Complete the rest of the look-up table shown below.

Decimal of Error Vector	Binary of Error Vector	Bit to be Flipped
0	0000	HAM_OK
1	0001	4
2	0010	5
3	0011	HAM_ERR

4	0100	6
5	0101	HAM_ERR
6	0110	HAM_ERR
7	0111	3
8	1000	7
9	1001	HAM_ERR
10	1010	HAM_ERR
11	1011	2
12	1100	HAM_ERR
13	1101	1
14	1110	0
15	1111	HAM_ERR

2. Decode the following codes. If it contains an error, show and explain how to correct it. Remember, it is possible for a code to be uncorrectable.
    a. $1110\ 0011_2$

$$\begin{matrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix} \quad * \quad \begin{matrix} 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \end{matrix}$$

$$= [1 \quad 2 \quad 3 \quad 3] \text{ (mod 2)} = [1 \quad 0 \quad 1 \quad 1]$$

- [1 0 1 1] Tells us that the error's at digit 2, so digit 2 must be flipped
    - [0, 1, 1, 1] -> flip bit 2 -> [0, 0, 1, 1]

    b. $1101\ 1000_2$

$$\begin{matrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{matrix}$$

$$
\begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \end{bmatrix}
$$

= [2 1 2 1] (mod 2) = [0, 1, 0, 1]

- [0, 1, 0, 1] = non-zero and not one of Ht's rows -> unfixable

NOTES from prelab Q's:

- When doing matrix multiplication, the (mod 2) means odds = 1, evens = 0
- Vector goes left -> right, binary goes right -> left