Bill Zhang
jzhan411@ucsc.edu
05/06/2021

CSE13S Spring 2021
Assignment 5: Hamming Codes
Design Document
This code will encode and decode information using Hamming codes to minimize errors.

Prelab Questions
1

0	Ok
1	4
2	5
3	Err
4	6
5	Err
6	Err
7	3
8	7
9	Err
10	Err
11	2
12	Err
13	1
14	0
15	Err

2a The hamming code multiplied by the Transposed parity check matrix is 0010. Flipped becomes 0100, which is 4. This means the error is the 6th bit according to the lookup table Flipping the 6th bit makes the code become 1110 0111.

2b The hamming code multiplied by the Transposed parity check matrix is 1010. Flipped becomes 0101, which is 9. This means the error is uncorrectable because the entry for 9 is Err.

Pseudocode

Bitvector
A bit vector is an ADT that represents a one dimensional array of bits, the bits in which are used to denote if something is true or false (1 or 0)

Structure bitvector
    Length (length in bits)
    Array *vector Array of bytes containing bits

BitVector *bv_create(uint32_t length)
The constructor for a bit vector. In the event that sufficient memory cannot be allocated, the function must return NULL.
Allocate memory for bv
If bv
    Set length
    allocate memory for vector
    Set each element of vector 0
    If didnt set vector
        Free bv
    Set v = null

void bv_delete(BitVector **v)
The destructor for a bit vector. Remember to set the pointer to NULL after the memory associated with the bit vector is freed.
    If v and if v->vector
        Free both
        *v = null

uint32_t bv_length(BitVector *v)
Returns the length of a bit vector.

void bv_set_bit(BitVector *v, uint32_t i)
Set ith bit in vit vector.

void bv_clr_bit(BitVector *v, uint32_t i)

Clears ith bit in bitvector

uint8_t bv_get_bit(BitVector *v, uint32_t i)
Returns ith bit

void bv_xor_bit(BitVector *v, uint32_t i, uint8_t bit)
Xors the ith bit in bit vector with value of specified bit

void bv_print(BitVector *v)
A debug function to print a bit vector.

Bit Matrix

Stricture
- Rows
- Columns
- Bitvector vector

BitMatrix *bm_create(uint32_t rows, uint32_t cols)
The constructor for a bit matrix. In the event that sufficient memory cannot be allocated, the function must return NULL. The
number of bits in the bit matrix is calculated as row × cols. Each bit should be initialized to 0.
Allocate memory for bm
If bm
- Set rows, cols
- Create vector using bv create
- If m vector
- Free m
- M = null

void bm_delete(BitMatrix **m)
The destructor for a bit matrix. Remember to set the pointer to NULL after the memory associated with the bit matrix is freed.
- If m and if m->vectort
- Free m
- Bv delete vector
- Set pointer to null;

uint32_t bm_rows(BitMatrix *m)
Returns the number of rows in the bit matrix.

uint32_t bm_cols(BitMatrix *m)
Returns the number of columns in the bit matrix.

void bm_set_bit(BitMatrix *m, uint32_t r, uint32_t c)
Sets the bit at row r and column c in the bit matrix.

void bm_clr_bit(BitMatrix *m, uint32_t r, uint32_t c)
Clears the bit at row r and column c .

uint8_t bm_get_bit(BitMatrix *m, uint32_t r, uint32_t c)
Gets the bit at row rand column c. Getting a bit should not modify any of the other bits.

BitMatrix *bm_from_data(uint8_t byte, uint32_t length)
Transforms first “length” amount of bits in byte to bitmatrix.
Create bm using bm create
    For each i in byte
    If i is 1
        Set bit in bm

bm_to_data(BitMatrix *m)
Extracts the first 8 bits of a bit matrix, returning those bits as a uint8_t.
Byte ; for each bit in vector set data bit to 1

BitMatrix *bm_multiply(BitMatrix *A, BitMatrix *B)
Performs a true matrix multiply, multiplying bit matrix A and bit matrix B mod 2.
Matrix multiplication code given

void bm_print(BitMatrix *m)
A debug function to print a bit matrix. You will want to do this first in order to verify the correctness of your bit matrix implementation.

Hamming Code

typedef enum HAM_STATUS {
HAM_OK = -3, // No error detected.
HAM_ERR = -2, // Uncorrectable.
HAM_CORRECT = -1 // Detected error and corrected.
} HAM_STATUS;

uint8_t ham_encode(BitMatrix *G, uint8_t msg)
Generates a Hamming code given a nibble of data stored in the lower of nibble of msg
and the generator matrix G. Returns the generated Hamming code, which is stored as a byte, or a Uint8_t.

Bm message = bmfromdata(message)
Bm hamming = multiply message and g
Data = bm to data
Delete message
Delete hamming
Return data

HAM_STATUS ham_decode(BitMatrix *Ht, uint8_t code, uint8_t *msg)
    Multiply code with the matrix ht, get error code
    Compared error code to look up table
    Fix errors if there are any
    Set msg to the correct message
    Return error code

Encode
Using a while opt function
    If case h print help message
    If case i, set infile to file
    If case o, set outfile to file

    Encode infile and set outfile to output

Decode
Using a while opt function
    If case h, print help message
    If case i, set infile to file
    If case o, set outfile to file
    If enable verbose printing of statistics.

I think I actually learned (or re-learned?) a ton of stuff during this lab. First and foremost is how bits work. While I think I did learn this in CSE12 along with Hamming codes in Linear Algebra, I had already forgotten most of it. The good news is that since I had already learned this before, it was easier for me to understand it for this lab. I learned how to set/clear/xor a bit and modify bits in a byte. The rest of the assignment was reviewed from previous assignments. I think the hardest part of this lab was actually the algorithm and logic behind the msb->lsb and transforming into lsb->msb. That took me a couple of hours of hours just to understand. Overall I thought that this lab was pretty fun, although a bit time consuming.