Devan O'Boyle
CSE13s
Assignment 5: Hamming Codes

Design

Purpose: For this lab, we will be encoding and decoding messages while using Hamming codes to correct errors caused by noise also known as random bit flips. In this scenario, we will be using the Hamming(8,4) code. because we will be taking in a nibble (4 bits) of data at a time and returning a Hamming code in a byte (8 bits) of data.

This assignment can be split up into two major parts: encoding and decoding.

Encoding:
For encoding, we take in a nibble (4 bits) of data at a time and return a byte (8 bits) of Hamming code. Generating a Hamming code will be done by using a given generator matrix as shown below:

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

By multiplying the identity matrix G by the given nibble and then taking the modulus 2 of it, we can calculate an 8 bit Hamming code. Note that we have to take the modulus of 2 so that the code remains in binary format (only 0's and 1's).

Pseudocode:
take the lower nibble of the byte
take the upper nibble of the byte
encode the lower nibble
encode the upper nibble
output the lower nibble and upper nibble
 in the encode function:
 convert the message to a bit matrix
 multiply the matrix by the generator
convert the code back to data
return the code

Decoding:
For decoding, we take a byte of Hamming code and multiply it by a parity checker matrix as shown:

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

Notice that the parity checker is the inverse of the generator matrix from before. This is what allows us to recover our message since we are basically doing the reverse of what we did when we encoded the message. Since we are taking in an 8 bit Hamming code and we want to get a 4 bit message, we are going to want to rearrange the parity checker matrix as shown:

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

This way, by multiplying an 8 bit wide matrix by a 4 bit wide matrix, we will return a 4 bit message. However, the decoder not only has to decode the message, but it also has to be able to check for errors in the message and correct the ones that it can. Therefore, a lookup will need to be performed by comparing the decoded message to that of the parity checker matrix to find which bit to flip. However, if the decoded message doesn't appear on the parity checker and the message wasn't correct, then there are too many errors and an error will be returned.

Pseudocode:
decode function:
convert the code to a bit matrix
multiply the code matrix and the checker matrix and set the product to an error matrix
iterate through the error and check if it contains only 0's
if so, then return HAM_OK to conclude that no errors were found
otherwise iterate through the checker matrix and compare line of bits with the error matrix
if the error matrix matches a line of bits, then return HAM_CORRECT to indicate that there was an error but it was corrected
otherwise if the error matrix didn't find a matching line of bits, return HAM_ERROR to indicate that there was more than one error and therefore the errors couldn't be corrected

In order to implement encoding and decoding, some structures will be needed to assist us in setting the bits and matrices that hold each set of bits. Therefore, two structures: one for a bit vector and one for the matrices of bits will be needed.

Bit Vector:
For the bit vector, a single byte will be taken in and set each of the eight bits into their own index in an array. There should be functions to create the bit vector and delete it, as with any structure, but in this case the create function should also initialize the bit vector to store an array of bits that are set to 0. The length of the array of bits should also be taken in and have a function that returns it's value for the corresponding bit vector. Then there are the functions that modify each of the bits. The set bit function will take the index of a bit and set that bit equal to 1. The clear bit function clears the bit by setting a corresponding bit to 0. The get bit function returns either a 1 or a 0 depending on what the value of the bit at that index is. There is also the xor function which will xor a bit with another, this is particularly useful when we want to multiply bits. Of course there is also a print function which is mainly used for debugging purposes, but it should iterate through the array of bits and print out each one.

Bit Matrix:
For the bit matrix, it takes in multiple bit vectors, and its main purpose is to perform the matrix multiplication for the encode and decode functions. The bit matrix will initialize its array of bit vectors in its create function. It will also have functions that return the number of rows and columns of the matrix. Of course, there is also a function to delete the matrix and free the memory allocated for it. The set, get, and clear bits all call on their corresponding bit vector functions by plugging in the corresponding row and column indices of the bit. By using the formula: row index * total number of columns in the matrix + column index, and plugging it into one of the bit vector functions as a parameter, the correct index of the bit in the matrix can be set, cleared, or returned. There is also the from data function that is a special kind of create function that is used specifically to read in data either in the form of a byte or a nibble and return a one dimensional matrix. There is also the to data function which takes the first 8 bits of a matrix and returns them as a byte. Then there is the multiply function which creates a new matrix and multiplies two other matrices together. The product of these matrices is then set to the newly created matrix which is then returned. Lastly, there is the print function which iterates through the bit matrix and prints out each set of bits.

For reading in input there are a variety of command line options. -i should be used to specify an input file and default to standard input. -o should be used to specify an output file and default to standard output. -h should display a help message, and for the decode function, -v should display various statistics about decode functions such as the number of bits read, the number of corrected and uncorrectable errors, and the ratio between the corrected and uncorrectable errors.

Pre-lab questions:

1.

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

2. a. 1110 0011, error code: 0010, 7th bit needs to be flipped
   -> 1110 0001

   b. 1101 1000, error code 1010, more than one bit needs to be flipped, therefore it is uncorrectable