Darren Lee
CSE13S
Spring 2021

Assignment 5: Hamming Codes
Design Document

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

0	0 or HAM_OK
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. 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) $1110\ 0011_2$

a) $\vec{e} = \vec{c} H^T = [1100 \ 0111] \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \pmod 2$

$= [1 \ 0 \ 1 \ 1]$

flip the 2nd bit because $[1 \ 0 \ 1 \ 1]$ is row 2 of H^T

$\vec{c} = [1000 \ 0111] = 1110\,0001_2$

$\vec{m} = [1 \ 0 \ 0 \ 0] = 0001_2$

c=[1100 0111]

$P_0=D_1$^D_2^D_3

0=/=1^0^0

$P_1=D_0$^D_2^D_3

1=1^0^0

$P_2=D_0$^D_1^D_3

1=/=1^1^0

$P_3=D_0$^D_1^D_2^D_3^P_0^P_1^P_2

1=/=1^1^0^0^0^1^1

P_0, P_2, and P_3 indicate the error. Since P_1 is correct, we know D_0, D_2, and D_3 are correct. That means D_1 is the flipped bit. We can flip D_1 to 0 to correct the error. This is error code 13, so flip the 2nd bit at index 1.

(b) $1101\ 1000_2$

b) $\vec{e} = \vec{c}\, H^T = [0001\ 1011] \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \pmod 2$

$= [0\ 1\ 0\ 1]$

This doesn't match any of the rows of H^T, so more than one bit has been flipped and we can't correct the error.

1=0^0^1
0=/=0^0^1
1=0^0^1
1=0^0^0^1^1^0^1
Only P_1 is wrong. However, if we flip P_1, we would also have to flip P_3. Therefore, this code is uncorrectable because more than one bit is flipped.

Purpose

The purpose of this lab is to create an encoder and a decoder. Using the (8,4) Hamming systematic code, the encoder will generate Hamming codes from input data, and the decoder will decode the generated Hamming codes. This program will allow us to perform error checking on data and correct those errors. If there is one error, the program can correct it and fix the message. If there is no error, the message does not need any correction. If there is more than one error, the program cannot correct them and does not modify the message.

Layout/Structure

- There is an encoder and decoder program that both use the Hamming (8,4) systematic code.
- The generator matrix G and the transpose of the parity-checker matrix H^T will be represented with bit matrices.
- The encoder program is called encode and supports the command line options -h, -i infile, and -o outfile.
- The decoder program is called decode and supports the command line options -h, -i infile, -o outfile, and -v.
- The provided error.c file injects errors into the Hamming codes.
- The encode.c file contains the Hamming Code encoder.
- The decode.c file contains the Hamming Code decoder.
- The error.c file contains the code to inject noise into the Hamming Codes.
- The entropy.c file contains the entropy measurement program.
- bv.h contains the bit vector ADT interface.
- bv.c contains the implementation of the bit vector ADT.
- bm.h contains the bit matrix ADT interface.
- bm.c contains the implementation of the bit matrix ADT.
- hamming.h contains the interface of the Hamming Code module
- hamming.c contains the implementation of the Hamming Code module.
- stat.h contains the declaration of the uncorrected_errors variable for counting how many errors are uncorrected when decoding.
- The Makefile can build, format and clean the program.
    - make and make all build the encoder, the decoder, the supplied error-injection program, and the supplied entropy-measure program.
    - make encode builds the decoder only
    - make decode builds the decoder only
    - make error builds the error-injection program only
    - make entropy builds the entropy-measure program only
    - make clean removes all compiler-generated files
    - make format formats the .c and .h files

Functionality

- The encoder program parses command-line options with getopt9) and opens input and output files with the correct file permissions. Then it creates a generator matrix G with bm_create(). After that, it reads a byte from the input file with fgetc(). Following that, it generates the Hamming(8,4) codes for the upper and lower nibbles using ham_encode() and writes to lower nibble then the upper nibble in the output file using fputc(). The read and write process is repeated until all the data has been read from the file. At the end, the input and output files are closed with fclose() and allocated memory is freed.
- The encode program supports the following command-line options:
    - -h: Prins out a help message that explains the purpose of the program and the command-line options it accepts, and then exits the program.
    - -i infile: Specifies the input file path that contains the data to encode into Hamming codes. The default input file is stdin

    - -o outfile: Specifies the output file path to write the encoded data (Hamming Codes). The default output file is stdout.
- The decode program first parses command line options with getopt() then opens input and output files with the correct file permissions with fopen(). Then the transpose parity-checker matrix H^T is created with bm_create(). The program then reads two bytes from the input file stream with fgetc(). The Hamming (8,4) codes are used to decode each byte pair read with ham_decde() to get the original upper and lower nibbles of the message and reconstruct the original byte. Then the reconstructed byte is written with fputc(). The read and write process is repeated until all the data has been read from the file. The number total bytes processed, number of uncorrected errors, number of corrected errors, and rate of uncorrected errors are printed to stderr with fprintf(). Finally, the input and output files are closed with fclose() and allocated memory is freed.
- The decode program supports the following command-line options:
    - -h: prints out a help message that explains the purpose of the program and the command-line options it accepts, then exits the program
    - -i infile: specifies the input file path containing Hamming codes to decode. The default input file is stdin.
    - -o outfile: Specifies the output file path to write the decoded Hamming Codes to. The default output file is stdout.
    - -v: prints the statistics of the decoding process to stderr. The number of bytes processed, number of uncorrected errors, number of corrected errors, and the error rate are printed. The error rate is the number of uncorrected errors/number of total bytes processed.
- The error.c program injects noise into the Hamming codes at a rate specified by -e rate (the default is 0.01 or 1%) between 0.0 and 1.0 inclusive. The seed is specified with -s seed and must be a positive integer.

  Error Handling:
- If the user enters an invalid command line option for encode or decode, the program will print the help and usage message and exit the program.
- If the infile or outfile fail to open in encode or decode, the program will print an error message to stderr that says the program failed to open the file and exit the program.

Pseudocode
bv.c
Define BYTE_SIZE 8

Struct code provided by Professor Long in assignment pdf
Struct BitVector{
        Int length
        8 bit int pointer vector
}

constructor function for bit vector influenced by Professor Long’s code on Piazza

```
BitVector *bv_create(int length):
	BitVector *v=(BitVector *)malloc(sizeof(BitVector))
	If (v):
		Int offset
		If (length % BYTE_SIZE==0):
			offset =0
		Else:
			offset=1
		v->vector=8 bit int pointer calloc(length/BYTE_SIZE+offset,sizeof(8 bit int))
		v->length=length
		If (!v->vector):
			Free v
			Set v to NULL
		Return v
	Else:
		Return NULL

Void bv_delete(BitVector **v):
	If (*v and *v vector)
		Free *v’s vector
		Free *v
		Set *v pointer to NULL

Int bv_length(BitVector *v):
	Return v->length

Code inspired by Eugene’s lab section on 5/4
Void bv_set_bit(BitVector *v, int i):
	v->vector[i/BYTE_SIZE] |= (1<<(i % BYTE_SIZE))

Code inspired by Eugene’s lab section on 5/4
Void bv_clr_bit(BitVector *v, int i):
	v->vector[i/BYTE_SIZE] &= ~(1<<(i%BYTE_SIZE))

Int bv_get_bit(BitVector *v, int i):
	Return (v->vector[i/BYTE_SIZE]) & (1<<(i%BYTE_SIZE))) >> (i % BYTE_SIZE)

Void bv_xor_bit(BitVector *v, int i, int bit):
	assert(bit<=1)
	v->vector[i/BYTE_SIZE] ^= (bit << (i % BYTE_SIZE))

Void bv_print(BitVector *v):
	For i in range bv_length(v):
		Print “<bv_get_bit(v,i)> ”
```

Print new line

bm.c
Define BYTE_SIZE 8

Struct code provided by Professor Long in assignment pdf
```
Struct BitMatrix {
	Int rows
	Int cols
	BitVector *vector
}

BitMatrix *bm_create(int rows, int cols):
	BitMatrix *m = (BitMatrix *) malloc(sizeof(BitMatrix))
	If (m):
		m->vector=bv_create(rows * cols)
		m->rows=rows
		m->cols=cols
		If (!m->vector):
			Delete m's vector
			Free m
			Set m to NULL
		Return m
	else:
		Return NULL

Void bm_delete(BitMatrix **m):
	If (*m and *m's vector):
		Delete *m's vector
		Free *m
		Set *m pointer to NULL

Int bm_rows(BitMatrix *m):
	Return m->rows

Int bm_cols(BitMatrix *m):
	Return m->cols

Void bm_set_bit(BitMatrix *m, int r, int c):
	bv_set_bit(m->vector,r*bm_cols(m)+c)

Void bm_clr_bit(BitMatrix *m, int r, int c):
	bv_clr_bit(m->vector, r*bm_cols(m)+c)
```

```
Int bm_get_bit(BitMatrix *m, int r, int c):
    Return bv_get_bit(m->vector, r*bm_cols(m)+c)

BitMatrix *bm_from_data(int byte, int length):
    assert(length<=BYTE_SIZE)
    BitMatrix *m =bm_create(1,length)
    For i in range length:
        If ((byte>>i)&1):
            bm_set_bit(m,0,1)
    Return m

Int bm_to_data(BitMatrix *m):
    8 bit int byte
    For i in range BYTE_SIZE:
        If (bm_get_bit(m,0,i)):
            byte |= (1<<i)
    Return byte

BitMatrix *bm_multiply(BitMatrix *A, BitMatrix *B):
    BitMatrix *m=bm_create(A->rows,B->cols)
    For i in range A->rows:
        For j in range B->cols:
            8 bit int sum=0
            For k in range A->cols:
                sum^=(bm_get_bit(A,i,k) & bm_get_bit(B,k,j)
        If (sum):
            bm_set_bit(m,i,j)
    Return m

Void bm_print(BitMatrix *m):
    For i in range bm_rows(m):
        For j in range bm_cols(m):
            Print <bm_get_bit(m,i,j)>
            If (j==bm_cols(m)-1):
                Print new line
```

hamming.c

```
Define TABLE_SIZE 16
Define NIBBLE_SIZE 4
Define BYTE_SIZE 8

Helper function provided by Professor Long in assignment pdf
8 bit int lower(8 bit int val):
    Return val & 0xF
```

Code influenced by Eugene’s lab section on 5/4
8 bit Int ham_encode(BitMatrix *G, 8 bit int msg):
    BitMatrix *m=bm_from_data(msg,NIBBLE_SIZE)
    BitMatrix *c=m*G
    8 bit int code = bm_to_data(c)
    Delete m
    Delete c
    Return code

Code influenced by Eugene’s lab section on 5/4
HAM_STATUS ham_decode(BitMatrix *Ht, 8 bit int code, 8 bit int *msg):
    Int lookup[TABLE_SIZE] = {
HAM_OK,4,5,HAM_ERR,6,HAM_ERR,HAM_ERR,3,7,HAM_ERR,HAM_ERR,HAM_ERR,2,HAM_ERR,1,0,HAM_ERR}
    BitMatrix *c=bm_from_data(code,BYTE_SIZE)
    BitMatrix *e=c*Ht
    8 bit int err=bm_to_data(e)
    Delete e
    If (err==0):
        *msg=lower(bm_to_data(c))
        Delete c
        Return HAM_OK
    If (lookup[err]==HAM_ERR):
        Increment uncorrected_errors
        Delete c
        Return HAM_ERR
    Else:
        If (bm_get_bit(c,0,lookup[err])):
            vm_clr_bit(c,0,lookup[err])
        Else:
            bm_set_bit(c,0,lookup[err])
        *msg=lower(bm_to_data(c))
        Delete c
        Return HAM_CORRECT

encode.c
Define OPTIONS “hi:o:”
Helper function provided by Professor Long in assignment pdf
8 bit int lower_nibble(8 bit int val):
    Return val & 0xF

9 bit int upper_nibble(8 bit int val):
    Return val >> 4

```
Code for statbuf and file permissions provided by Professor Long in assignment pdf
Int main(int argc, char **argv):
	Struct stat statbuf
	Declare int opt and int c
	FILE *infile=stdin
	FILE *outfile=stdout
	While ((opt=getopt(argc,argv,OPTIONS))!=-1):
		switch(opt):
		Case h:
			Print help message
			Return 0
		Case i:
			infile =open optarg with reading permission
			Break
		Case o:
			outfile =open optarg with writing permission
			Break
		Default:
			Print help message
			Return 1
	If (infile == NULL):
		Print error message to stderr
		Return 1
	If (outfile == NULL):
		Print error message to stderr
		Return 1
	fstat(fileno(infile),&statbuf)
	Fchmod(fileno(outfile,statbuf.st_mode)
	Create generator matrix G
	While ((c=fgetc(infile))!=EOF):
		Print ham_encode(G,lower_nibble(c) to outfile
		Print ham_encode(G,upper_nibble(c) to outfile
	Close infile
	Close outfile
	Delete G
	Return 0
```

decode.c

```
Define OPTIONS “hi:o:v”

Helper function provided by Professor Long in assignment pdf
8 bit int pack_byte(8 bit int upper, 8 bit int lower):
	Return (upper<<4) | (lower & 0xF)
```

Code for statbuf and file permissions provided by Professor Long in assignment pdf

```
Int main(int argc, char **argv):
    Struct stat statbuf
    Int bytes_processed=0
    Int corrected_errors=0
    Int opt, c_low=0, stat=0
    FILE *infile=stdin, *outfile=stdout
    While ((opt=getopt(argc,argv,OPTIONS))!=-1):
        switch(opt):
        Case h:
            Print help message
            Return 0
        Case i:
            infile=open optarg for reading
            break
        Case o:
            Outfile=open optarg for writing
            break
        Case v:
            stat=1
            Break
        Default:
            Print help message
            Return 1
    If (infile == NULL):
        Print error message to stderr
        Return 1
    If (outfile == NULL):
        Print error message to stderr
        Return 1
    fstat(fileno(infile),&statbuf)
    fchmod(fileno(outfile),statbuf.st_mode)
    Create H transpose matrix Ht
    While ((c_low=fgetc(infile))!=EOF):
        HAM_STATUS status_low=ham_decode(Ht,c_low,&msg_low)
        If (status_low==HAM_CORRECT):
            Increment corrected_errors
        If (status_high==HAM_CORRECT):
            Increment corrected errors
        Bytes_processed +=2
        Print pack_byte(msg_high,msg_low) to outfile
    If (stat):
        Print statistics (bytes_processed,uncorrected_errors,corrected_errors,error rate)
```

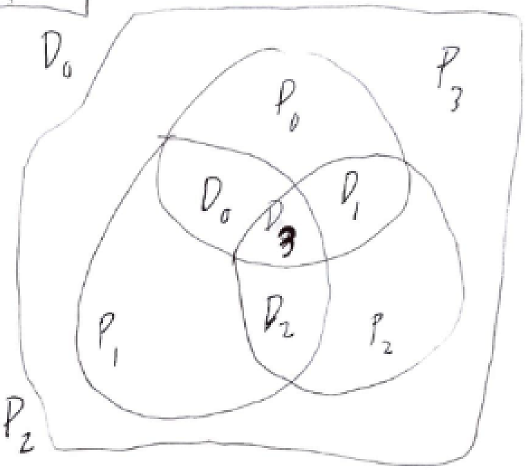
    Close infile
    Close outfile
    Return 0

stat.h

Declare int uncorrected_errors

Draft Work

1	0	0	1	1	1	0	0
P_3	P_2	P_1	P_0	D_3	D_2	D_1	D_0



$P_0 = D_0 \wedge D_1 \wedge D_3$

$P_1 = D_0 \wedge D_2 \wedge D_3$

$P_2 = D_1 \wedge D_2 \wedge D_3$

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

$$G = \begin{array}{c} \begin{array}{cccccccc} P_3 & P_0 & P_1 & D_2 & & & & \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

Bit Matrix has m rows, n columns

To set r^{th} row, c^{th} column

$r \times n + c$

Decoding ( )

code
c = bm_ from_data

$\vec{e} = c \cdot H^T$

if e is 0
 no error, return HAM_OK

else
 lookup(e)

low = read byte

high = read next byte

decode lower.

decode upper

fputc(pack byte (upper, lower))