Mattiwos Belachew
mbelache@ucsc.edu
5/9/2021

CSE 13s Spring 2021
Assignment 5:
Hamming Codes
Design Document

Prelab:

1.

1.)

error (MSB … LSB)	row in H^T (0-index)
0000	HAM_OK
0001	4
0010	5
0011	HAM_Err
0100	6
0101	HAM-Err
0110	Ham_Err
0111	3
1000	7
1001	Ham_Err
1010	Ham_Err
1011	2
1100	Ham_Err
1101	1
1110	0
1111	Ham_Err

2.

a.

2.)

a.) 1110 0011

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

$P_0 = D_1 \wedge D_2 \wedge D_3$

$= 1 \wedge 0 \wedge 0 = 1 \neq \overset{\overset{0}{\shortparallel}}{P_0}$

$P_1 = D_0 \wedge D_2 D_3$

$= 1 \wedge 0 \wedge 0 = 1 = P_1 = 1$

$P_2 = D_0 \wedge D_1 \wedge D_3$

$= 1 \wedge 1 \wedge 0 = 0 = P_2 \neq 1$

$P_3 = D_0 \wedge D_1 \wedge D_2$

$= 1 \wedge 1 \wedge 0 = 0 = P_3 \neq 1$

• All errors occurs because of D_1 so it has to be flipped to 0

Correct = $\underbrace{1110}_{P}$ $\underbrace{0001}_{D}$

b.

6.) $1101\ 1000_2$

$P_0 = D_1 \wedge D_2 \wedge D_3$ $\quad$ $P_0 = 1$

$= 0 \wedge 0 \wedge 1 = 1 \equiv P_0$

$P_1 = D_0 \wedge D_2 D_3$

$= 0 \wedge 0 \wedge 1 = \underline{1} \not\equiv P_1$ $\quad$ $P_1 = 0$

$P_2 = D_0 \wedge D_1 \wedge D_3$

$= 0 \wedge 0 \wedge 1 = \underline{1} \equiv P_2$ $\quad$ $P_2 = 1$

$P_3 = D_0 \wedge D_1 \wedge D_2$

$= 0 \wedge 0 \wedge 0 = 0 \not\equiv P_3$ $\quad$ $P_3 = 1$

- This isn't correctable because both P_1 & P_3 are wrong, and they have D_0 and D_2 in common. But P_2 & P_0 is correct so no possible replacement.

Description:

For this assignment, I will be implementing hamming codes, encoding and decoding them. It used to counteract noisy interference, because transferring data through a noisy communication channel is prone to errors. This is done by adding extra information to our data which allows us to perform error checking, and request that the sender retransmit any data that was incorrect. In addition, not only detect eros but we can also correct them. This technique is called forward error correction (FEC).

TOP LEVEL:

Encode:

```
encode{
    opt = 0;
    output = stdout;
    input = stdin;
    C = 0;

    while ((opt = getopt(argc, argv, OPTIONS)) != -1) { //program argument
parser.
        switch (opt) {
        case 'h':
             //print message
            exit(1);
            break;
        case 'i': input = fopen(optarg, "rb"); break;
        case 'o': output = fopen(optarg, "wb"); break;
        }
    }

    //create Generator Matrix G
    while ((c = fgetc(input)) != EOF){
         //read byte
         //encode top nibble
         //put it in the output
```

```
        //encode bottom nibble
        //put it in the output
    }

    //close program
    //delete G bit matrix
    //close input and output
   //return successful exit
}
```

Decode:

```
decode(int argc, char **argv) {
    opt = 0;
   output = stdout;
    input = stdin;
    verbose = false;
    C = 0x0;

    while ((opt = getopt(argc, argv, OPTIONS)) != -1) { //program argument
parser.
        switch (opt) {
        case 'h':
             //print message
            exit(1);
            break;
        case 'i': input = fopen(optarg, "rb"); break;
        case 'o': output = fopen(optarg, "wb"); break;
         case 'v': verbose = true; break;
        }
    }

    //create Generator Matrix H 4x8

    while ((byte = fgetc(input)) != EOF){
```

```
        //read byte
        //decode byte

         //check return HAM type and add it to its corresponding counter
         //read byte
        //decode byte
         //check return HAM type and add it to its corresponding counter

        //combine both msg values together and push it to output

    }
    if (verbose){
        printf("PRINT DATA \n");
    }
    //close program
    //delete H bit matrix
    //close input and output files
    //return successful exit
}
```

BitVector:

```
BitVector *bv_create(uint32_t length){
    //create bitvector
    assert(v); //check if it created
    v.length = length;
      v.vector = calloc((length/8) + length%8 * 8,sizeof(uint8_t));
    if (v.vector){
        free(v);
      v = NULL;
        return NULL;
    }
    return v;
}
```

```
bv_delete(BitVector **v){
    assert(*v);
    assert((*v).vector);
    free((*v).vector);
    free(*v);
    return;
}
bv_length(BitVector *v){
    return v.length;
}
bv_set_bit(BitVector *v, i){
    v.vector[i/8] |= (0x1<<(i%8));//or to set
    return;
}
bv_clr_bit(BitVector *v, i){
    v.vector[i/8] &= ~(0x1<<(i%8));//or to set
    return;

}
bv_xor_bit(BitVector *v,  i, bit){
    v.vector[i/8] ^= (bit<<(i%8));//xor to set
    return;
}
bv_get_bit(BitVector *v, i){
    return (v.vector[i/8]&(0x1 <<(i%8)))>>(i%8);

}
bv_print(BitVector *v){
    for (i = v.length; i >= 0 ;i--){
        printf(bv_get_bit(v,i));
    }
    printf("\n");
    return;
}
```

BitMatrix:

```
BitMatrix *bm_create(rows, cols) {
    BitMatrix *m = (BitMatrix *) calloc(1, sizeof(BitMatrix));
    //check if m worked
    m->rows = rows;
    m->cols = cols;
    m->vector = bv_create(rows * cols);
    //check if m->vector worked
    return m;
}

bm_delete(BitMatrix **m) {
    //check if m exists
    bv_delete(&(*m)->vector);
    free(*m);
    *m = NULL;
    return;
}

bm_rows(BitMatrix *m) {
    //check if m exists
    return m->rows;
}

bm_cols(BitMatrix *m) {
    //check if m exists
    return m->cols;
}

bm_set_bit(BitMatrix *m, uint32_t r, uint32_t c) {
    //check if m exists
    bv_set_bit(m->vector, (r * m->cols) + c);
}

bm_clr_bit(BitMatrix *m, uint32_t r, uint32_t c) {
    //check if m exists
    bv_clr_bit(m->vector, (r * m->cols) + c);
}
```

```
bm_get_bit(BitMatrix *m, uint32_t r, uint32_t c) {
    //check if m exists
    return bv_get_bit(m->vector, (r * m->cols) + c);
}

BitMatrix *bm_from_data(uint8_t byte, uint32_t length) {
    BitMatrix *c = bm_create(1, length);
    for loop though length {
        if (byte & (1 << i)) {
            bm_set_bit(c, 0, i);
        } else {
            bm_clr_bit(c, 0, i);
        }
    }
    return c;
}

bm_to_data(BitMatrix *m) {
    //check if m exists
    //start at c = 0
    for loop through the cols) {
        c |= (bm_get_bit(m, 0, i)) << i; //shift to its corresponding position
    }
    return c;
}

BitMatrix *bm_multiply(BitMatrix *A, BitMatrix *B) {
    BitMatrix *C = bm_create(bm_rows(A), bm_cols(B));
     //rows in a{
        //cols in B{
             val = 0;
            //a cols{
                val ^= bm_get_bit(A, i, k) & bm_get_bit(B, k, j);
            }
            if (val % 2) {//mod 2
                bm_set_bit(C, i, j);
            }
        }
    }
    return C;
```

```
}

bm_print(BitMatrix *m) {
    for (loop through rows) {
        for (loops through cols) {
            printf("%u ", bm_get_bit(m, i, j));
        }
        //print new line
    }
}
```

Hamming:

```
lookup[16] = { HAM_OK, 4, 5, HAM_ERR, 6, HAM_ERR, HAM_ERR, 3, 7,
HAM_ERR, HAM_ERR, 2, HAM_ERR, 1, 0, HAM_ERR };
encode(G, msg){
        //c = bm_from_data(msg,4)
        //code = bm_multiply c,G
        //f = bm_to_data(code)
        //remove all bm and free them
        Return f
}
Decode (Ht, code, msg){
        BitMatrix *c = bm_from_data(code, 8);
        //bm_print(c);
        BitMatrix *err = bm_multiply(c, Ht);
        errormsg = bm_to_data(err);
        //loopup(errormsg)
        //If ham ok
                // msg = c
                //return HAM_OK
        //Else if ham_error
                //msg = c
                //return HAM_ERR
        //else
                //flip bitmatrix c at location lookup(errormsg)
                //msg = c
                //return HAM_CORRECT
}
```

Design Process:

1. In the beginning I was trying to implement a 2d bit matrix but I learned that it was wrong and I need to make a 1d matrix which acts as a 2d matrix which I found interesting and a lot easier than the method I was trying to implement.
2. I had a hard time with decode in the beginning because I wasn't aware I had two read two bytes at a time since encodes multiplied original space by two since it combined a nibble with another nibble which contains information to figure out the original message if an error occurs. I think this method of encoding and decoding might be a bit enficent since multiplying the original message by 2 in order to prevent information loss during transfer. I hope there is a better method out there since it doesn't seem to be too good in my opinion.

What I learned:

1. I learned how to manipulate bits in c and implement a bit matrix and learned how to implement a bit matrix multiplication.
2. I learned what a hamming code is and the use of it is.
3. I learned what entropy is.

Resources Used:

- Asgn5.pdf
- asgn5Design by William Santosa from Piazza
- Example_design.pdf found in CSE 13s Discord.