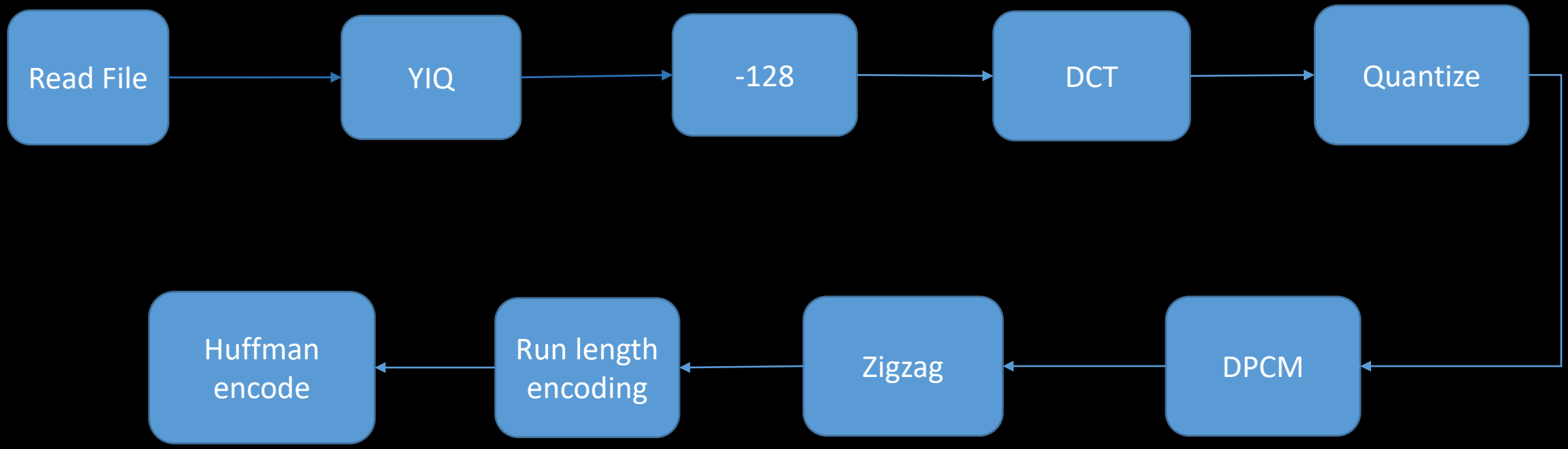
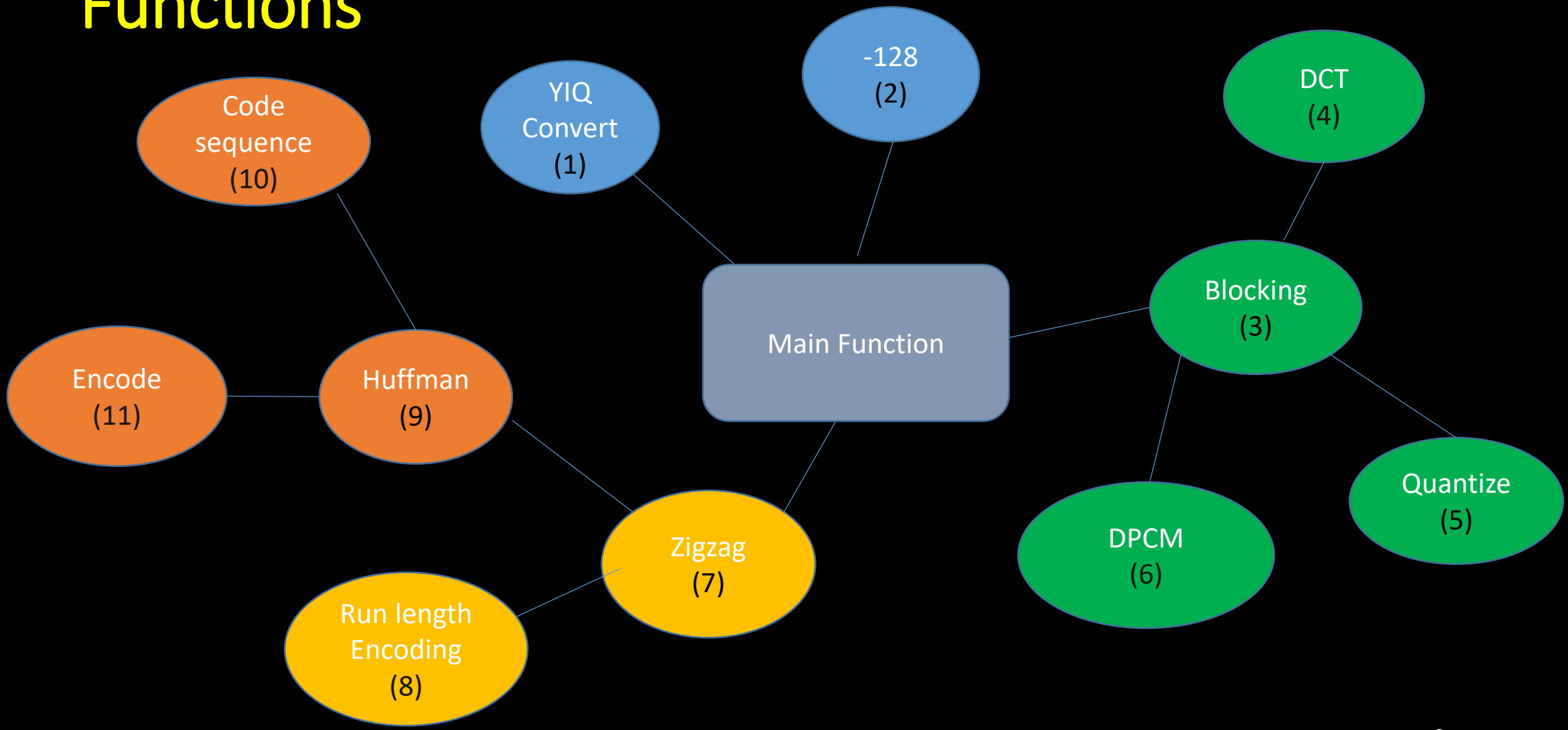
Multimedia Signal Processing Final Project

410686034 通訊三 徐陽瑄

Steps


Read File
YIQ
-128
DCT
Quantize
DPCM
Zigzag
Run length encoding
Huffman encode

Functions


Code sequence (10)
YIQ Convert (1)
-128 (2)
DCT (4)
Encode (11)
Huffman (9)
Main Function
Blocking (3)
Run length Encoding (8)
Zigzag (7)
DPCM (6)
Quantize (5)

Step 1: Convert YIQ

Y:Luminance I:In-phase Q: Quadrature-phase

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.595 & -0.274 & -0.322 \\ 0.211 & -0.522 & 0.312 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

```
for(i=0;i<row;i++){
    for(j=0;j<col;j++){
        YIQ[i][j].Y = 0.299*RGB[i][j].R + 0.587*RGB[i][j].G + 0.114*RGB[i][j].B;
        YIQ[i][j].I = 0.596*RGB[i][j].R - 0.274*RGB[i][j].G - 0.322*RGB[i][j].B;
        YIQ[i][j].Q = 0.211*RGB[i][j].R - 0.523*RGB[i][j].G + 0.312*RGB[i][j].B;
    }
}
```

Step 2: -128 (Shift Value)

Subtract every value by 128

```
void shift_value(ImgYIQ **YIQ, int row, int col){
    int x,y;
    for(x=0;x<row;x++){
        for(y=0;y<col;y++){
            YIQ[x][y].Y = YIQ[x][y].Y-128.0;
            YIQ[x][y].I = YIQ[x][y].I-128.0;
            YIQ[x][y].Q = YIQ[x][y].Q-128.0;
        }
    }
}
```

Step 3:Blocking

Process every 8*8 matrix in the picture

```
for(i=0;i<block_h;i++){
    for(j=0;j<block_w;j++){
        for(k=0;k<8;k++){
            for(h=0;h<8;h++){
                DFT_data[k][h].Y = data[i*8+k][j*8+h].Y;
                DFT_data[k][h].I = data[i*8+k][j*8+h].I;
                DFT_data[k][h].Q = data[i*8+k][j*8+h].Q;
            }
        }
```

Functions after blocking goes here
DCT -> Quantize

Step 4:DCT

$$F(u,v) = \sum_{r=0}^{M-1} \sum_{S=0}^{N-1} \frac{2C(u)C(v)}{\sqrt{MN}} f(r,s) \cos\left(\frac{(2r+1)u\pi}{2M}\right) \cos\left(\frac{(2s+1)v\pi}{2N}\right)$$

$$C(\delta) = \frac{\sqrt{2}}{2} \;\; when \;\; \delta = 0 \;\;\; otherwise \;\;\; C(\delta) = 1$$

```
for(u=0;u<8;u++){
    for(v=0;v<8;v++){
        if(u==0){ Cu=0.707106781; }
        else{ Cu=1.0; }
        if(v==0){ Cv=0.707106781; }
        else{ Cv=1.0;}
        //float x = sqrt(u*v);
        for(r=0;r<8;r++){
            for(s=0;s<8;s++){
                x+=data[r][s].Y*(cos(((2*r)+1)*u*PI/16))*(cos(((2*s)+1)*v*PI/16));
                y+=data[r][s].I*(cos(((2*r)+1)*u*PI/16))*(cos(((2*s)+1)*v*PI/16));
                z+=data[r][s].Q*(cos(((2*r)+1)*u*PI/16))*(cos(((2*s)+1)*v*PI/16));
            }
        }
        pixel[u][v].Y = Cu*Cv*x/4;
        pixel[u][v].I = Cu*Cv*y/4;
        pixel[u][v].Q = Cu*Cv*z/4;
        x=0.0;
        y=0.0;
        z=0.0;
    }
}
```

Step 6:Quantize

Quantize table for Y and I, Q

```
int quantize_table_Y[8][8] = {
    {16, 11, 10, 16, 24, 40, 51, 61},
    {12, 12, 14, 19, 26, 58, 60, 55},
    {14, 13, 16, 24, 40, 57, 69, 56},
    {14, 17, 22, 29, 51, 87, 80, 62},
    {18, 22, 37, 56, 68,109,103, 77},
    {24, 35, 55, 64, 81,104,113, 92},
    {49, 64, 78, 87,103,121,120,101},
    {72, 92, 95, 98,112,100,103, 99}
};
```

```
int quantize_table_IQ[8][8] = {
    {17,18,24,47,99,99,99,99},
    {18,21,26,66,99,99,99,99},
    {24,26,56,99,99,99,99,99},
    {47,66,99,99,99,99,99,99},
    {99,99,99,99,99,99,99,99},
    {99,99,99,99,99,99,99,99},
    {99,99,99,99,99,99,99,99},
    {99,99,99,99,99,99,99,99}
};
```

Step 7:DPCM

Subtract the DC value with the previous block but only preserve the DC value of the hole picture.

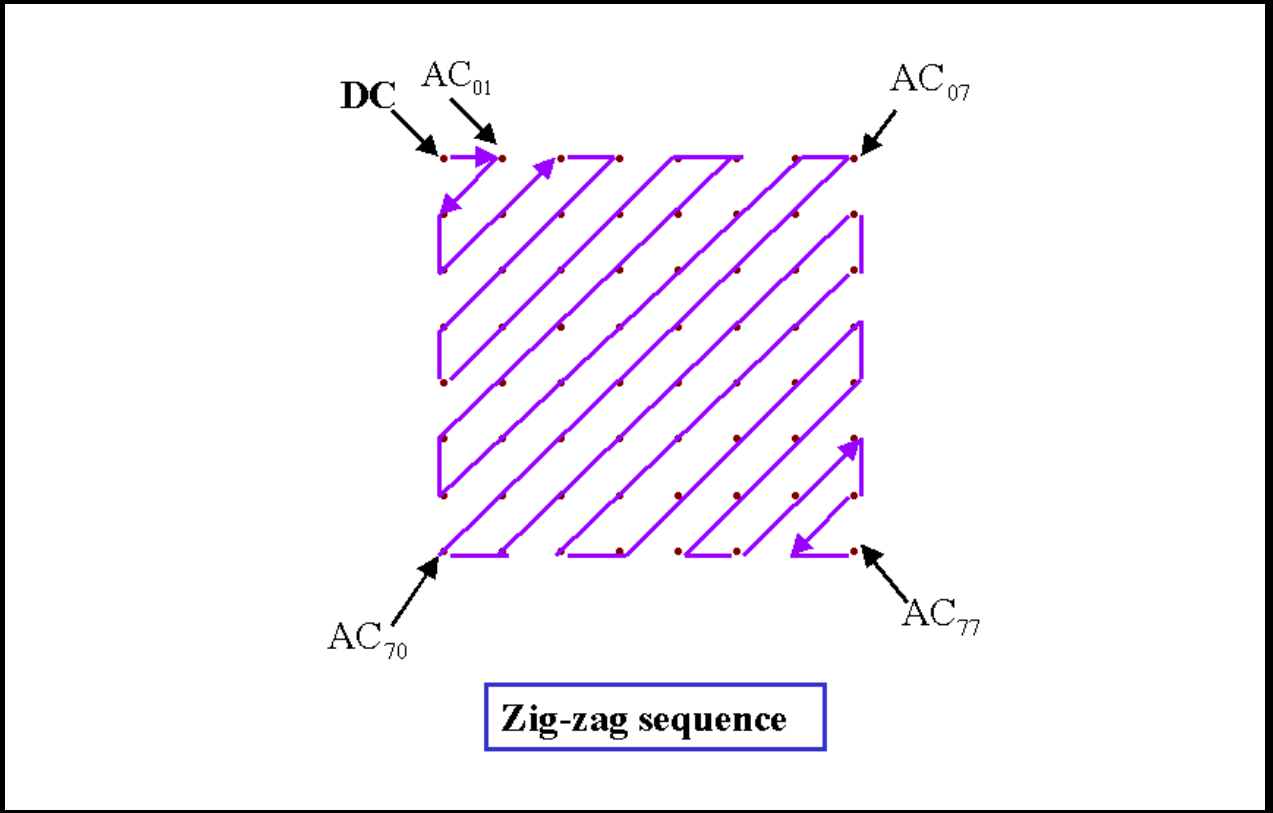
Save the DC value of the first block

Subtract the value with the previous block

```
if(i==0 && j==0){
    first_DC.Y = quant[0][0].Y;
    printf("first_DC_Y = %d ", first_DC.Y);
    first_DC.I = quant[0][0].I;
    printf("first_DC_I = %d ", first_DC.I);
    first_DC.Q = quant[0][0].Q;
    printf("first_DC_Q = %d \n", first_DC.Q);
}else{
    pre_Y = quant[0][0].Y;
    pre_I = quant[0][0].I;
    pre_Q = quant[0][0].Q;
    quant[0][0].Y = DC_dif(quant[0][0].Y, first_DC.Y);
    quant[0][0].I = DC_dif(quant[0][0].I, first_DC.I);
    quant[0][0].Q = DC_dif(quant[0][0].Q, first_DC.Q);
    first_DC.Y = pre_Y;
    first_DC.I = pre_I;
    first_DC.Q = pre_Q;
}
```

Step 8:Zigzag + RLE

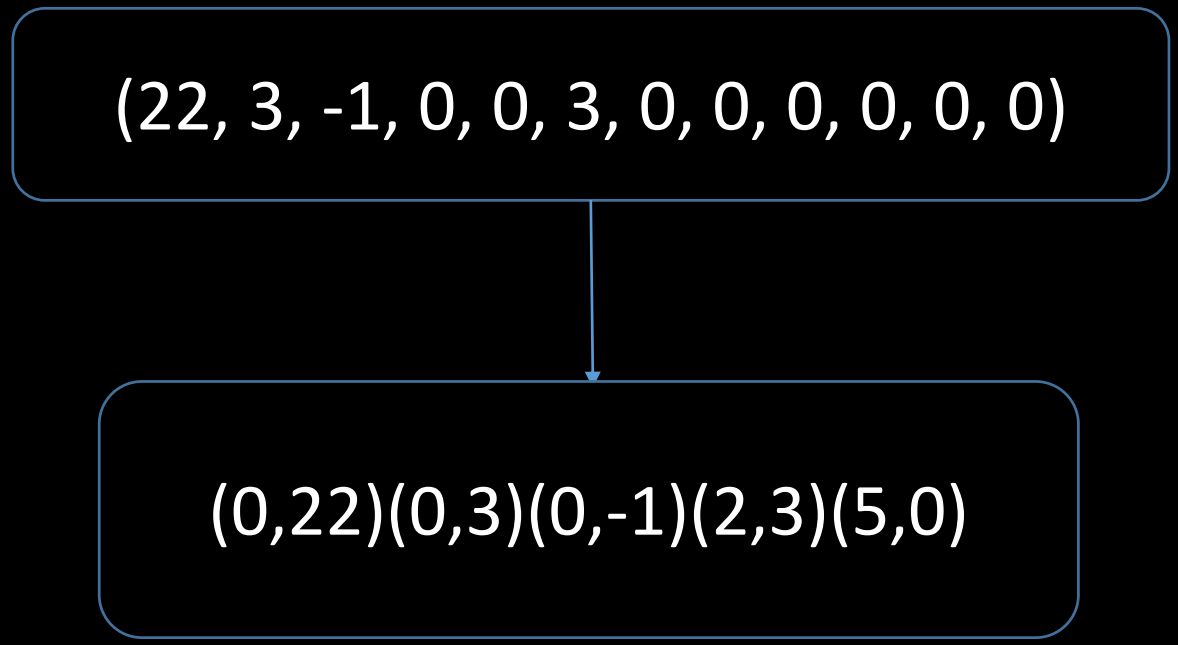
Zigzag : To save the data in certain pattern. Saving in this pattern will increase the numbers of continuous zeros, hence, while doing RLE, the compression would be better.



Step 8:RLE

A method for compression, counting the amount of zeros before any number that does not equal to zero.

EX.



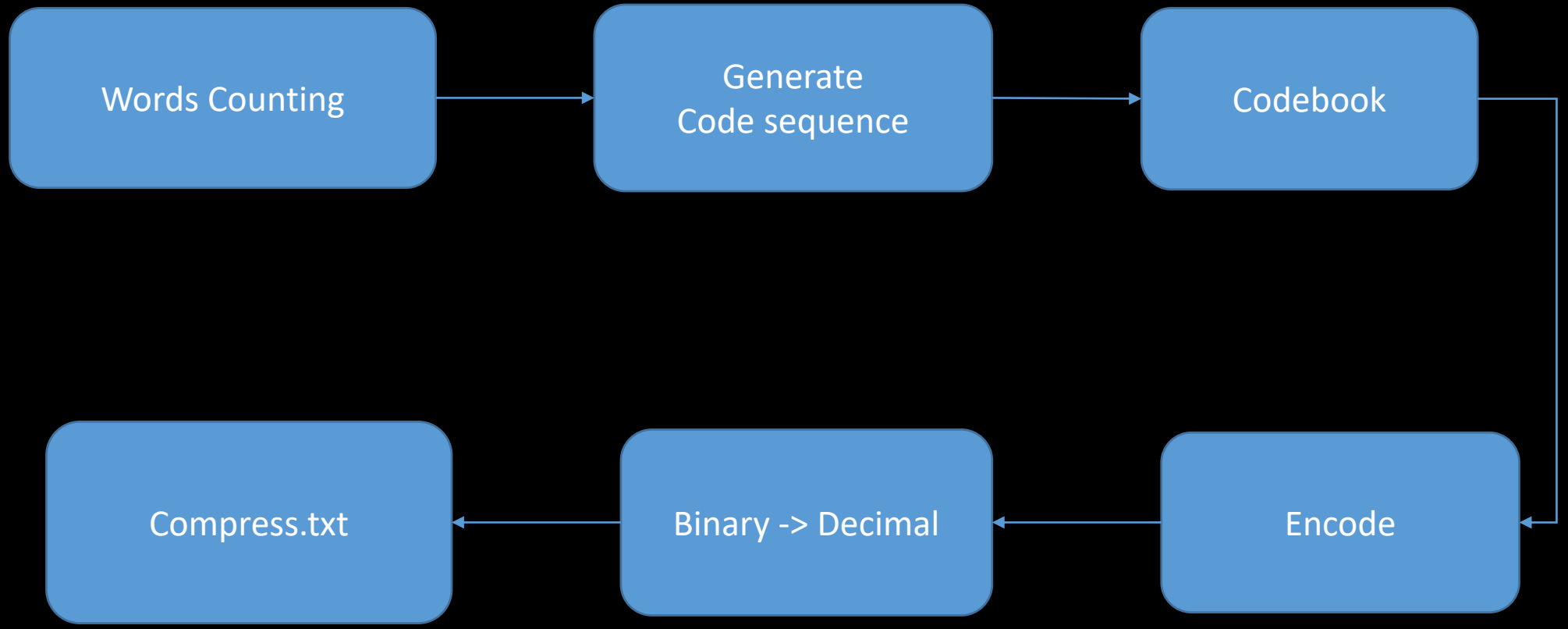
```
int x[]={0,0,1,2,1,0,0,1,2,3,4,3,2,1,0,0,1,2,3,4,5,6,5,4,3,2,1,0,0,1,2,3,4,5,6,7,7,6,5,
        4,3,2,1,2,3,4,5,6,7,7,6,5,4,3,4,5,6,7,7,6,5,6,7,7};
int y[]={0,1,0,0,1,2,3,2,1,0,0,1,2,3,4,5,4,3,2,1,0,0,1,2,3,4,5,6,7,6,5,4,3,2,1,0,1,2,3,
        4,5,6,7,7,6,5,4,3,2,3,4,5,6,7,7,6,5,4,5,6,7,7,6,7};
for(i=0;i<height/8;i++){
    for(j=0;j<width/8;j++){
        for(k=0;k<8;k++){
            for(h=0;h<8;h++){
                num[k][h].Y = data[i*8+k][j*8+h].Y;
                num[k][h].I = data[i*8+k][j*8+h].I;
                num[k][h].Q = data[i*8+k][j*8+h].Q;
            }
        }

        if(num[x[k]][y[k]].Y != 0){
            *(zigzag_Y+Y) = flag_Y;
            *(zigzag_Y+Y+1) = num[x[k]][y[k]].Y;
            Y=Y+2;
            flag_Y=0;
        }else{
            flag_Y++;
        }
```

Note:
I and Q do the same thing

Using flag to count the numbers of zeros

Step 9:Huffman Steps



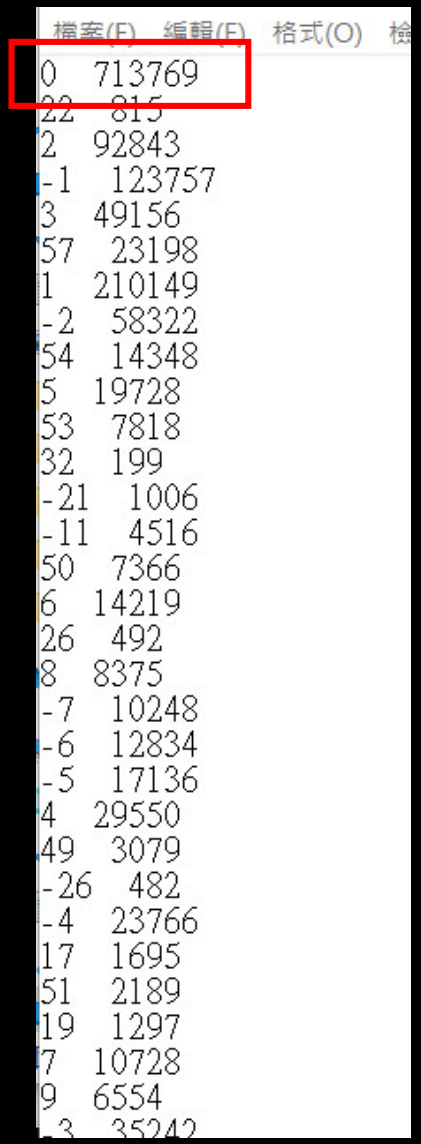
Step 9-1:Count words

```
for(i=1;i<size;i++){
    for(j=0;j<flag;j++){
        if(data[i] == zig_data[j].num){
            zig_data[j].count++;
            append=1;
            break;
        }
    }
    if(append!=1){
        zig_data[flag].num = data[i];
        zig_data[flag].count = 1;
        flag++;
        append=0;
    }else{
        append=0;
    }
}
```

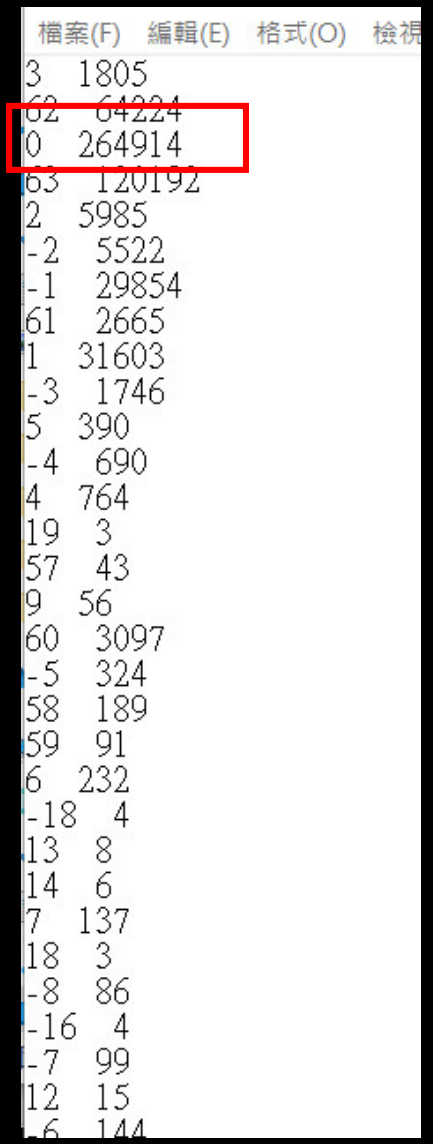
Saving different values into a structure, simultaneously counting the numbers of the value.

```
typedef struct data_zig{
    int num;
    int count;
    int code[30];
    int len;
} data_zig;
```

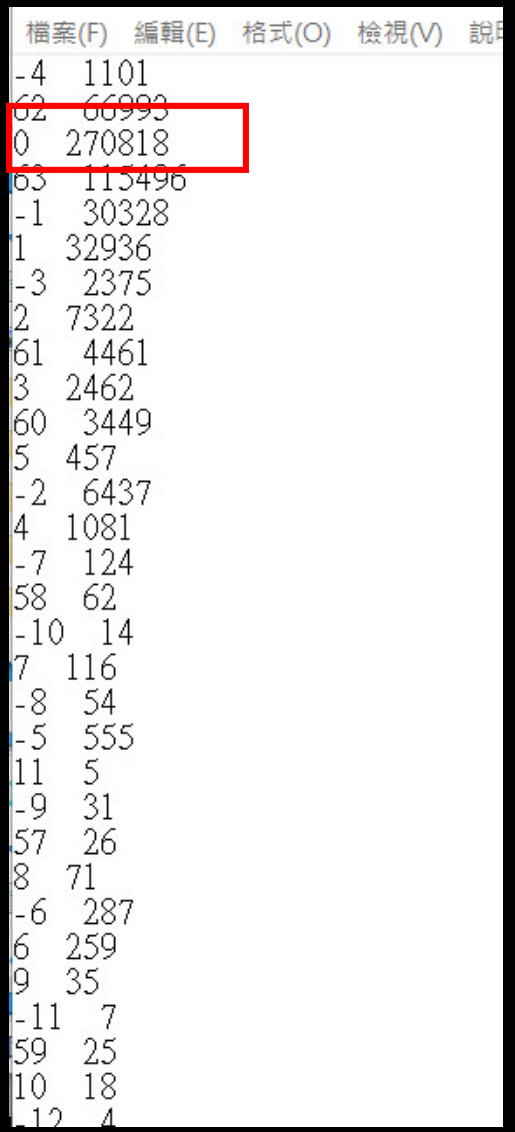
Y Output



I Output



Q Output



Step 9-2: Code sequence

To get the position of every time when we add x[0] and x[1]

```
code sequence generating...
code sequence length = 1714450
12108642014121086499121089121311121212121114131312111110121110121111131316151720201919201919182019191819191919212221
21202020222121222322222223232222232423222123222122212423222222212021202020191819181919201919191918182019181717181717171615151514141313131211111098876655432l0
```

```
code sequence generating...
code sequence length = 535185
864201197510898648775676556776776776655544432343332101l110
```

```
code sequence generating...
code sequence length = 547415
10221244444443233323213343211111l0
```

Step 9-3:Codebook

Y Codebook

9	00101101
-9	00110000
55	00111101
50	00111110
53	01001100
-8	01001110
8	01001111
-7	01101100
7	01101110
58	01110110
-6	0010111
6	0011001
63	0011010
54	0011011
-5	0110000
61	0110100
5	0110101
57	0111010
-4	001010
62	001110
4	010010
-3	011001
60	011100
3	00100
-2	01000
2	01111
-1	0101
1	000
0	1

I Codebook

-12	111101100011100
-10	111101111100010
10	111101111101100
57	111101111101101
-9	111101100011111
9	11110111101010
8	11110111110000
-8	11110111110111
59	1111011000110
-7	1111011110100
7	1111011111001
-6	1111011111010
58	111101100010
6	111101111011
-5	11110110000
5	11110111100
-4	11110111111
4	1111011001
-3	111101101
3	111101110
61	11110010
60	11110011
-2	1111000
2	1111010
-1	11111
1	1110
62	110
63	10
0	0

Q Codebook

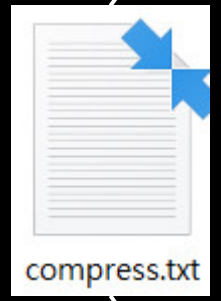
-11	1111110111111100011
-10	111111011111110000
10	111111011111110001
59	111111011111111001
57	111111010010000
-9	111111010010001
9	11111101111001
-8	111111011111101
58	11111101001001
8	1111110111101
7	1111110111111
-7	111111010100101
6	111111010011
-6	111111011110
5	11111101000
-5	111111011110
4	1111110101
-4	1111110110
-3	111111110
3	111111111
60	11111100
61	11111110
-2	1111100
2	1111101
-1	11110
1	1110
62	110
63	10
0	0

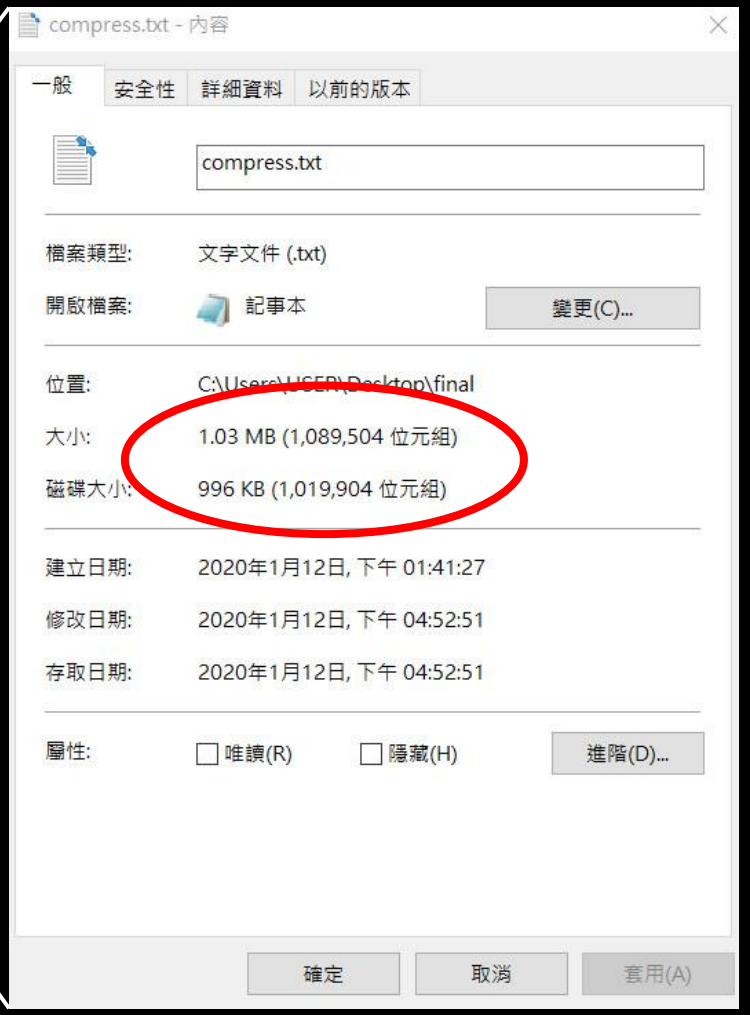
Step 9-4:Binary->Decimal

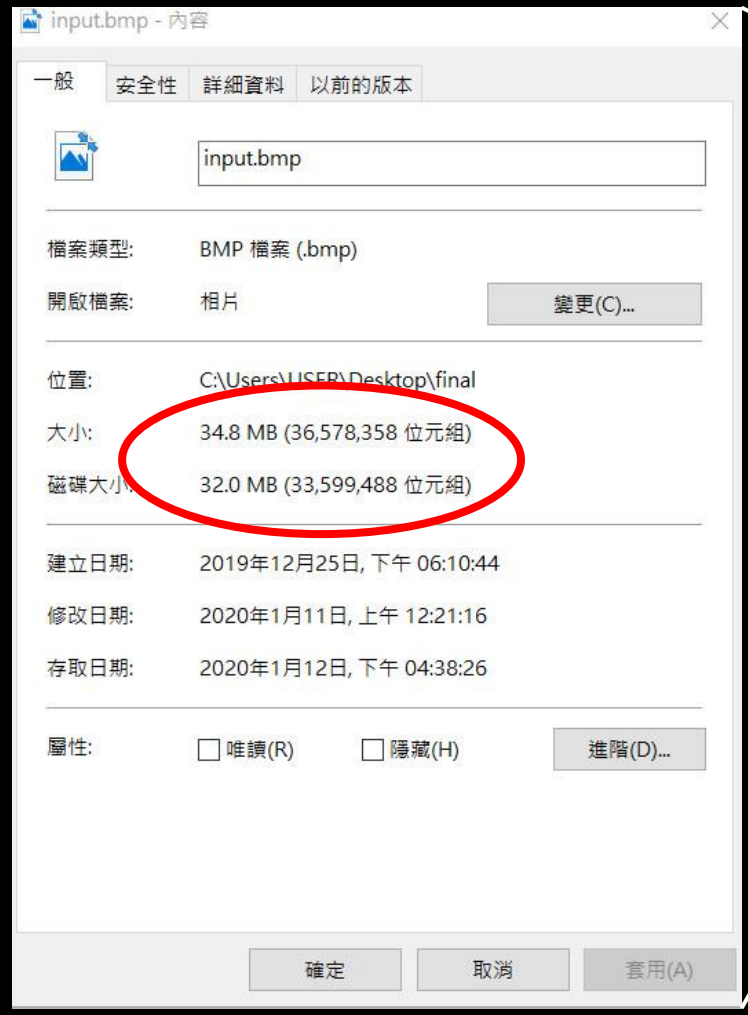
```
for(i=0;i<size_Y/8;i++){
    for(j=0;j<8;j++){
        temp[j] = data_Y[i*8+j];
    }
    total =(temp[0])*128 + (temp[1])*64 + (temp[2])*32 + (temp[3])*16 + (temp[4])*8 + (temp[5])*4 + (temp[6])*2 + (temp[7]);
    fprintf(fp, "%c", total);
}
```

I and Q is similar as Y

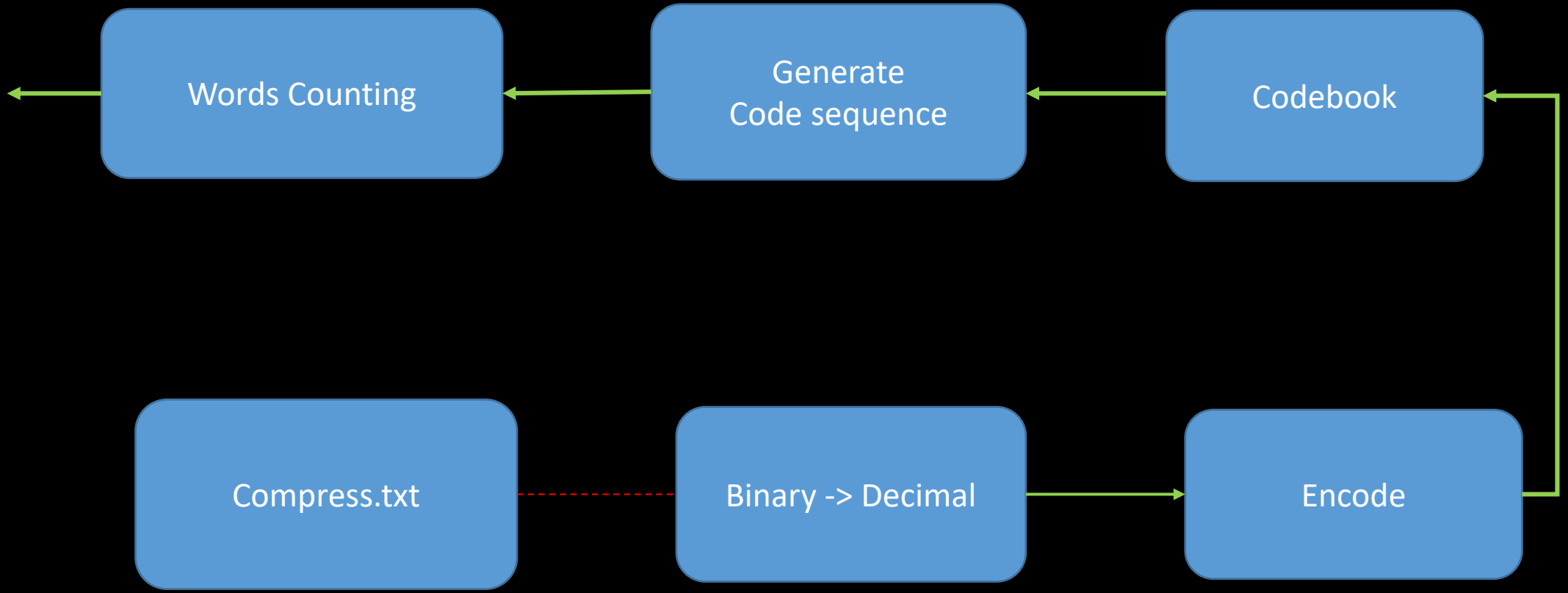
Step 9-5:Encode



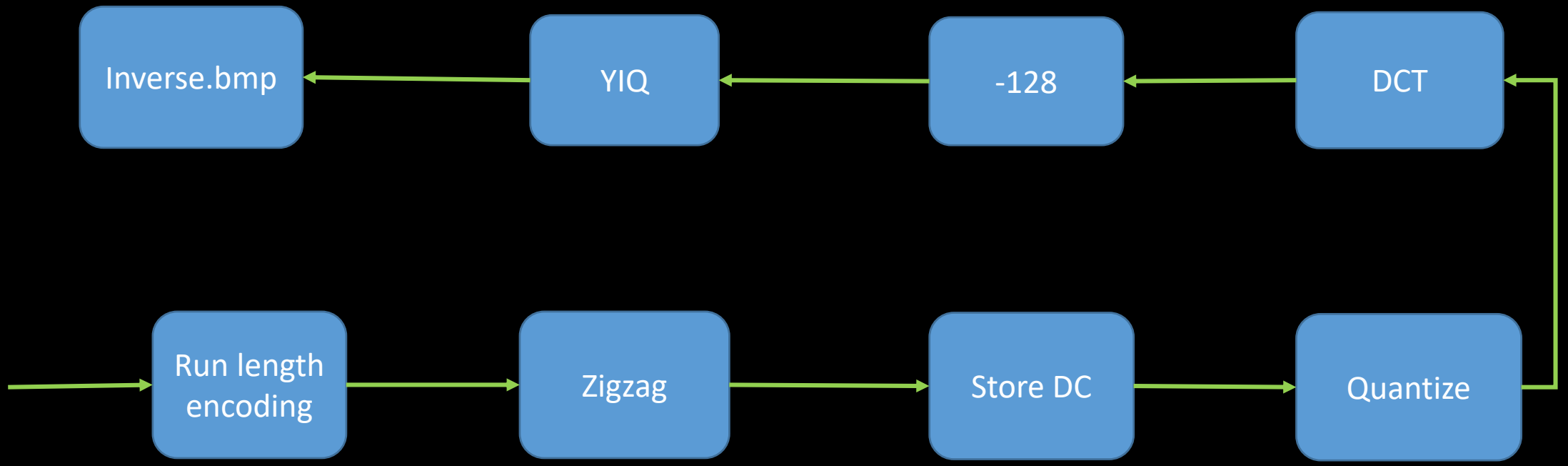




Inverse


Words Counting
Generate
Code sequence
Codebook
Compress.txt
Binary -> Decimal
Encode

Inverse


Inverse.bmp
YIQ
-128
DCT
Run length encoding
Zigzag
Store DC
Quantize

Inverse

Inverse.bmp

Input.bmp

參考資料:

- https://en.wikipedia.org/wiki/YIQ
- https://en.wikipedia.org/wiki/JPEG
- https://crazycat1130.pixnet.net/blog/post/1345538
- http://nova.bime.ntu.edu.tw/~ttlin/Course01/download_files/C1TECH_DOC_04.pdf