信道编码实验报告

一、汉明码——线性代数观点

1. C1——（15,11）汉明码两个矩阵的推导过程

$$H=\begin{array}{c}\begin{array}{cccccccccccccc}1&2&3&4&5&6&7&8&9&10&11&12&13&14&15\end{array}\\ \begin{pmatrix}1&0&1&0&1&0&1&0&1&0&1&0&1&0&1\\0&1&1&0&0&1&1&0&0&1&1&0&0&1&1\\0&0&0&1&1&1&1&0&0&0&0&1&1&1&1\\0&0&0&0&0&0&0&1&1&1&1&1&1&1&1\end{pmatrix}\end{array}$$

$$H_S=\begin{array}{c}\begin{array}{cccccccccccccc}C:1&2&3&4&5&6&7&8&9&10&11&12&13&14&15\\3&5&6&7&9&10&11&12&13&14&15&1&2&4&8\end{array}\\ \begin{pmatrix}1&1&0&1&1&0&1&0&1&0&1&1&0&0&0\\1&0&1&1&0&1&1&0&0&1&1&0&1&0&0\\0&1&1&1&0&0&0&1&1&1&1&0&0&1&0\\0&0&0&0&1&1&1&1&1&1&1&0&0&0&1\end{pmatrix}\end{array}=(A|I_4)$$

$$G_S=\begin{array}{c}\begin{array}{cccccccccccccc}1&2&3&4&5&6&7&8&9&10&11&12&13&14&15\end{array}\\ \begin{pmatrix}1&0&0&0&0&0&0&0&0&0&0&1&1&0&0\\0&1&0&0&0&0&0&0&0&0&0&1&0&1&0\\0&0&1&0&0&0&0&0&0&0&0&0&1&1&0\\0&0&0&1&0&0&0&0&0&0&0&1&1&1&0\\0&0&0&0&1&0&0&0&0&0&0&1&0&0&1\\0&0&0&0&0&1&0&0&0&0&0&0&1&0&1\\0&0&0&0&0&0&1&0&0&0&0&1&1&0&1\\0&0&0&0&0&0&0&1&0&0&0&0&0&1&1\\0&0&0&0&0&0&0&0&1&0&0&1&0&1&1\\0&0&0&0&0&0&0&0&0&1&0&0&1&1&1\\0&0&0&0&0&0&0&0&0&0&1&1&1&1&1\end{pmatrix}\end{array}$$

H: $C_{5\sim11}\to C_{4\sim15}$, $C_{12\sim15}\to C_{5\sim8}$

$C_{12}\leftrightarrow C_{5\sim6}$, $C_3\leftrightarrow C_5$, $C_4\leftrightarrow C_7$, $C_5\leftrightarrow C_6$

C_1：最终列顺序：12 13 1 14 2 3 4 15 5 6 7 8 9 10 11（指序的标号，非列内容）

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

2. 以矩阵乘法编程实现（15,11）汉明码

（1）代码实现

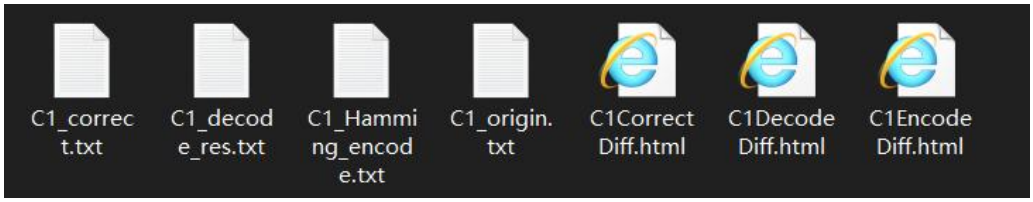
使用 python 编程，调用 numpy 实现矩阵运算。具体代码如下所示。

```python
def C1encode(s:int):
    tmp = bin(s)[2:].zfill(11)[::-1]
    d = [[]]
    for i in range(11):
        d[0].append(int(tmp[i]))
    res = np.dot(d, G)
    out = ''
    for x in res[0]:
        out += str(x % 2)
    return out
```

```python
def C1decode(r: str):
    R = str2list(r)
    z = np.dot(H, R)
    tmp = list2int(z)
    if tmp != 0:
        R[tmp-1][0] = ~R[tmp-1][0] % 2
    r = ''
    for i in range(15):
        r += str(R[i][0])
    res = r[2]+r[4:7]+r[8:]
    return res
```

（2）编码结果验证

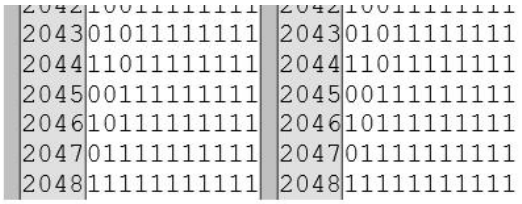
C1 验证中生成的所有中间文件见 codes/C1，截图如下：



编码结果验证代码如下。采用上一节编写的程序生成一个和 hamming_15_11.txt 格式相同的文件，再比较两个文件的内容。

```python
def vrfy_encode():
    with open('C1_Hamming_encode.txt', 'w') as f:
        for i in range(2048):
            # i = bin(i)[2:].zfill(11)[::-1]
            tmp = C1encode(i)
            # f.writelines(i+', '+tmp+'\n')
            f.writelines(bin(i)[2:].zfill(11)[::-1] + ', ' + tmp + '\n')
    f1 = open('C1_Hamming_encode.txt', 'r')
    f2 = open('hamming_15_11.txt', 'r')
    txt1 = f1.read().splitlines()
    txt2 = f2.read().splitlines()
    d = difflib.HtmlDiff()
    htmlContent = d.make_file(txt1,txt2)
    print(htmlContent)
    with open('C1EncodeDiff.html','w') as f3:
        f3.write(htmlContent)
    f1.close()
    f2.close()
```

验证结果见 codes/C1/C1EncodeDiff.html，部分验证结果如下截图所示，两文件无差异。



2043	01011111111	2043	01011111111
2044	11011111111	2044	11011111111
2045	00111111111	2045	00111111111
2046	10111111111	2046	10111111111
2047	01111111111	2047	01111111111
2048	11111111111	2048	11111111111

Legends

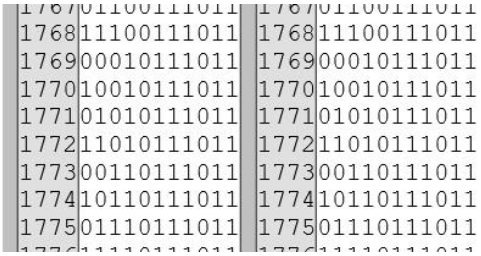
Colors	Links
Added	(f)irst change
Changed	(n)ext change
Deleted	(t)op

（3）解码结果验证

编码结果验证代码如下。生成两个文件：C1_origin.txt 存原始待编码数据，C1_decode_res.txt 存 C1 编码程序结果对应的解码结果。

```
def vrfy_decode():
    f1 = open('C1_origin.txt', 'w')
    f2 = open('C1_decode_res.txt', 'w')
    for i in range(2048):
        # i = bin(i)[2:].zfill(11)[::-1]
        f1.writelines(bin(i)[2:].zfill(11)[::-1]+'\n')
        r = C1encode(i)
        d = C1decode(r)
        f2.writelines(d + '\n')
    f1.close()
    f2.close()
    f1 = open('C1_origin.txt', 'r')
    f2 = open('C1_decode_res.txt', 'r')
    txt1 = f1.read().splitlines()
    txt2 = f2.read().splitlines()
    d = difflib.HtmlDiff()
    htmlContent = d.make_file(txt1, txt2)
    print(htmlContent)
    with open('C1DecodeDiff.html', 'w') as f3:
        f3.write(htmlContent)
    f1.close()
    f2.close()
```

验证结果见 codes/C1/C1DecodeDiff.html，部分验证结果如下截图所示，两文件无差异，解码结果正确。



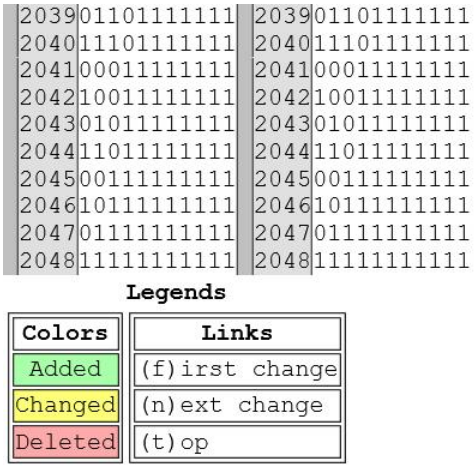
1768	11100111011	1768	11100111011
1769	00010111011	1769	00010111011
1770	10010111011	1770	10010111011
1771	01010111011	1771	01010111011
1772	11010111011	1772	11010111011
1773	00110111011	1773	00110111011
1774	10110111011	1774	10110111011
1775	01110111011	1775	01110111011

（4）纠正一位错验证

纠正一位错验证代码如下。将每个编码结果随机改变一位，再解码。生成一个文件：C1_correct.txt 存对存在一位错误的解码结果。C1_decode_res.txt 是上一节中生成的无差错时正确解码结果。

```
def vrfy_correct():
    f = open('C1_correct.txt','w')
    for i in range(2048):
        # i = bin(i)[2:].zfill(11)[::-1]
        r = C1encode(i)
        w = randint(0, 14)
        if r[w] == '0':
            r = r[0:w] + '1' + r[w+1:]
        else:
            r = r[0:w] + '0' + r[w + 1:]
        d = C1decode(r)
        f.writelines(d + '\n')
    f.close()
    f1 = open('C1_correct.txt', 'r')
    f2 = open('C1_decode_res.txt', 'r')
    txt1 = f1.read().splitlines()
    txt2 = f2.read().splitlines()
    d = difflib.HtmlDiff()
    htmlContent = d.make_file(txt1, txt2)
    print(htmlContent)
    with open('C1CorrectDiff.html', 'w') as f3:
        f3.write(htmlContent)
    f1.close()
    f2.close()
```

验证结果见 codes/C1/C1CorrectDiff.html，部分验证结果如下截图所示，两文件无差异，成功纠正一位错。



2039	01101111111	2039	01101111111
2040	11101111111	2040	11101111111
2041	00011111111	2041	00011111111
2042	10011111111	2042	10011111111
2043	01011111111	2043	01011111111
2044	11011111111	2044	11011111111
2045	00111111111	2045	00111111111
2046	10111111111	2046	10111111111
2047	01111111111	2047	01111111111
2048	11111111111	2048	11111111111

Legends

Colors	Links
Added	(f)irst change
Changed	(n)ext change
Deleted	(t)op

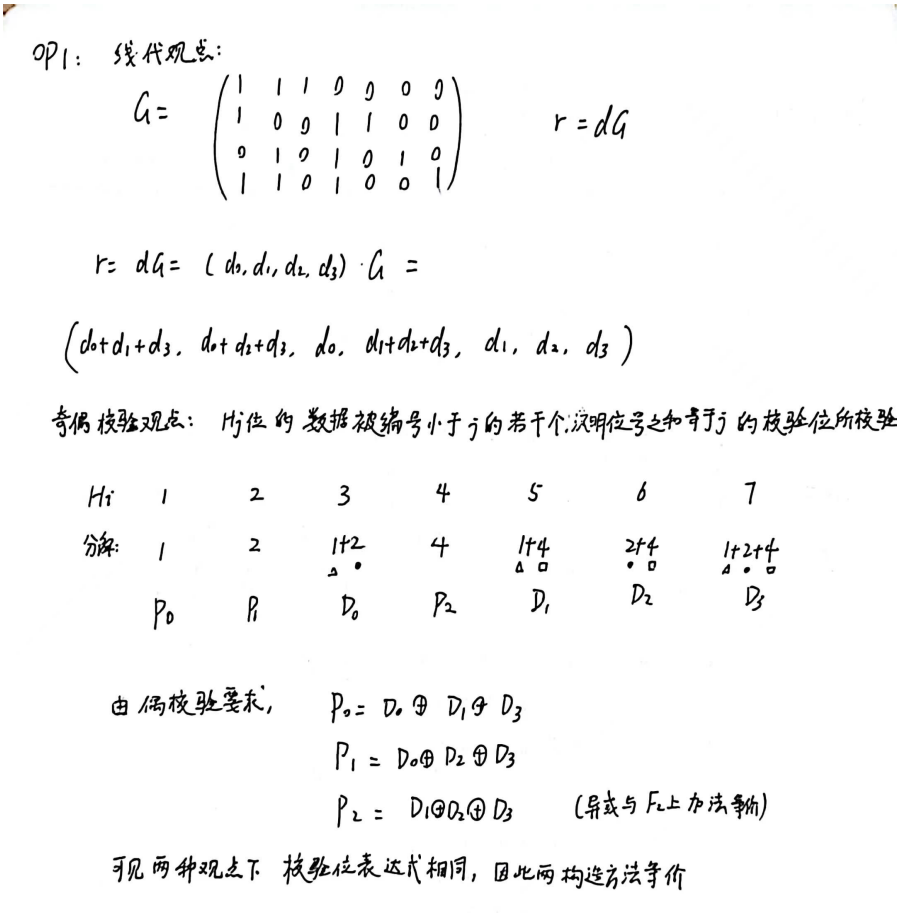
3. C2——使用矩阵乘法的问题

使用矩阵乘法进行汉明编码要进行一次矩阵乘法运算，对于（255,247）汉明码要进行247*255 次乘法，246*255 次加法，对于运算资源有限的嵌入式设备来说无法一次并行运算，即使是一行乘一列的操作也需要拆分成多批次，导致运算时间过长。根据矩阵乘法的运算法则，要先算乘法再算加法，而乘法器速度比加法器慢，导致并行效果不好，加法器时间利用率低。译码也是要进行一次矩阵乘法，但是计算量小一些，问题没有那么严重。

另一方面，如果要进行多次编码的话，就必须存储完整的生成矩阵 G，所需的存储空间太大，使得存中间结果的寄存器减少。为了等待寄存器空闲，计算的并行程度大大降低了，计算时间增加。 假如该嵌入式设备存储极度紧张，只能存下 G，那用原来的方法就无法展开计算。或许可以考虑每次编码都重新传一遍 G，一次传一列，计算完一行乘一列的结果以后再传入下一列，这样空间消耗会小一点，但是多次传数据增加的时间也可能比不能并行计算而增加的时间更长，那样就得不偿失了。

二、汉明码——奇偶校验观点

1. OP1——证明奇偶校验编码方法与线性代数构造方法等价



2. C3——使用优化编码译码方法实现（15,11）汉明码

（1）代码实现

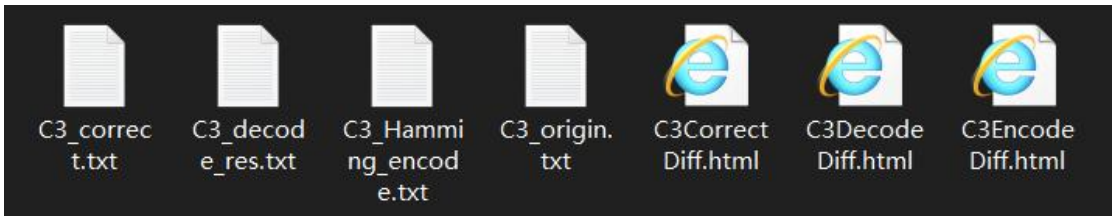
使用优化方法，运用一些字符串拼接操作避开矩阵运算。

```python
def C3encode(d: str):
    D = '0'+'0'+d[0]+'0'+d[1:4]+'0'+d[4:]
    b = 0
    for i in range(15):
        if D[i] == '1':
            b ^= i+1
    p = bin(b)[2:].zfill(4)[::-1]
    r = p[0] + p[1] + d[0] + p[2] + d[1:4] + p[3] + d[4:]
    return r
```

```python
def C3decode(r: str):
    b = 0
    for i in range(15):
        if r[i] == '1':
            b ^= i+1
    if b != 0:
        if r[b-1] == '0':
            r = r[0:b-1] + '1' + r[b:]
        else:
            r = r[0:b-1] + '0' + r[b:]
    d = r[2]+r[4:7]+r[8:]
    return d
```

（2）编码结果验证

C3 验证中生成的所有中间文件见 codes/C3，截图如下：



编码结果验证代码如下。采用上一节编写的程序生成一个和 hamming_15_11.txt 格式相同的文件，再比较两个文件的内容。

```python
def vrfy_encode():
    with open('C3_Hamming_encode.txt', 'w') as f:
        for i in range(2048):
            i = bin(i)[2:].zfill(11)[::-1]
            tmp = C3encode(i)
            f.writelines(i+', '+tmp+'\n')
    f1 = open('C3_Hamming_encode.txt', 'r')
    f2 = open('hamming_15_11.txt', 'r')
    txt1 = f1.read().splitlines()
    txt2 = f2.read().splitlines()
    d = difflib.HtmlDiff()
    htmlContent = d.make_file(txt1,txt2)
    print(htmlContent)
    with open('C3EncodeDiff.html','w') as f3:
        f3.write(htmlContent)
    f1.close()
    f2.close()
```

验证结果见 codes/C3/C3EncodeDiff.html，部分验证结果如下截图所示，两文件无差异。

2043	01011111111	2043	01011111111
2044	11011111111	2044	11011111111
2045	00111111111	2045	00111111111
2046	10111111111	2046	10111111111
2047	01111111111	2047	01111111111
2048	11111111111	2048	11111111111

Legends

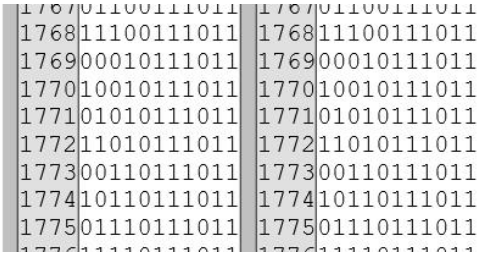
Colors	Links
Added	(f)irst change
Changed	(n)ext change
Deleted	(t)op

（5）解码结果验证

编码结果验证代码如下。生成两个文件：C3_origin.txt 存原始待编码数据，C3_decode_res.txt 存 C3 编码程序结果对应的解码结果。

```python
def vrfy_decode():
    f1 = open('C3_origin.txt', 'w')
    f2 = open('C3_decode_res.txt', 'w')
    for i in range(2048):
        i = bin(i)[2:].zfill(11)[::-1]
        f1.writelines(i+'\n')
        r = C3encode(i)
        d = C3decode(r)
        f2.writelines(d + '\n')
    f1.close()
    f2.close()
    f1 = open('C3_origin.txt', 'r')
    f2 = open('C3_decode_res.txt', 'r')
    txt1 = f1.read().splitlines()
    txt2 = f2.read().splitlines()
    d = difflib.HtmlDiff()
    htmlContent = d.make_file(txt1, txt2)
    print(htmlContent)
    with open('C3DecodeDiff.html', 'w') as f3:
        f3.write(htmlContent)
    f1.close()
    f2.close()
```

验证结果见 codes/C3/C3DecodeDiff.html，部分验证结果如下截图所示，两文件无差异，解码结果正确。



（6）纠正一位错验证

纠正一位错验证代码如下。将每个编码结果随机改变一位，再解码。生成一个文件：C3_correct.txt 存对存在一位错误的解码结果。C3_decode_res.txt 是上一节中生成的无差错时正确解码结果。

```python
def vrfy_correct():
    f = open('C3_correct.txt','w')
    for i in range(2048):
        i = bin(i)[2:].zfill(11)[::-1]
        r = C3encode(i)
        w = randint(0, 14)
        if r[w] == '0':
            r = r[0:w] + '1' + r[w+1:]
        else:
            r = r[0:w] + '0' + r[w + 1:]
        d = C3decode(r)
        f.writelines(d + '\n')
    f.close()
    f1 = open('C3_correct.txt', 'r')
    f2 = open('C3_decode_res.txt', 'r')
    txt1 = f1.read().splitlines()
    txt2 = f2.read().splitlines()
    d = difflib.HtmlDiff()
    htmlContent = d.make_file(txt1, txt2)
    print(htmlContent)
    with open('C3CorrectDiff.html', 'w') as f3:
        f3.write(htmlContent)
    f1.close()
    f2.close()
```

验证结果见 codes/C3/C3CorrectDiff.html，部分验证结果如下截图所示，两文件无差异，成功纠正一位错。

2039	01101111111		2039	01101111111	
2040	11101111111		2040	11101111111	
2041	00011111111		2041	00011111111	
2042	10011111111		2042	10011111111	
2043	01011111111		2043	01011111111	
2044	11011111111		2044	11011111111	
2045	00111111111		2045	00111111111	
2046	10111111111		2046	10111111111	
2047	01111111111		2047	01111111111	
2048	11111111111		2048	11111111111	

Legends

Colors	Links
Added	(f)irst change
Changed	(n)ext change
Deleted	(t)op

3. C4——奇偶校验观点的优势

奇偶校验没有用到矩阵乘法，因此规避了 C2 中提到的关于矩阵乘法的缺点，因此明显的优点就是时间上，由于不用大量计算乘法加法而缩短了编码译码时间；空间上，不用存储巨大的生成矩阵、校验矩阵而节省了存储资源，间接减少了运算速度。并且采用奇偶校验观点不用乘法器，只要加法器就能完成运算，缩短了运算时间，节省了运算资源。

三、汉明码——直观体会

1. C5——标准输入输出

修改了编码译码的输入来源，从标准输入读入数据，分组处理后再标准输出。分别保存为 my_encode.py 和 my_decode.py。

```
def Encode():
    d = input()
    l = len(d)
    for i in range(0, l, 11):
        r = C3encode(d[i:11+i])
        sys.stdout.write(r)
```

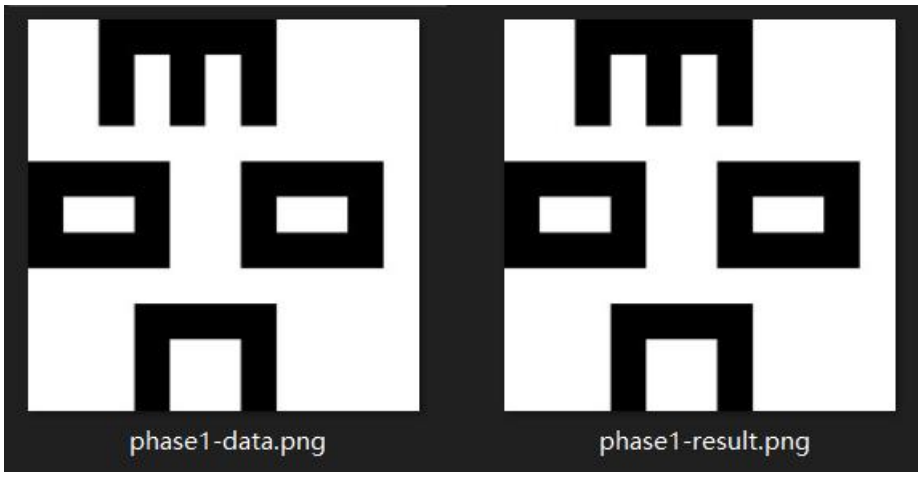
```
def Decode():
    r = input()
    l = len(r)
    for i in range(0, l, 15):
        d = C3decode(r[i:15 + i])
        sys.stdout.write(d)
```

2. C5.1——标准

在 Windows 下运行命令结果：

```
H:\THINGS\study\专业课\信息论与编码\实验\Hamming\codes>lab2_program_windows_amd64.exe -mode=source -phase=1 | python my_
encode.py | lab2_program_windows_amd64.exe -mode=channel -phase=1 | python my_decode.py | lab2_program_windows_amd64.exe
 -mode=verify -phase=1
Phase 1: Verify ok.
Your images: phase1-data.png, phase1-result.png
```

生成的可视化图片：



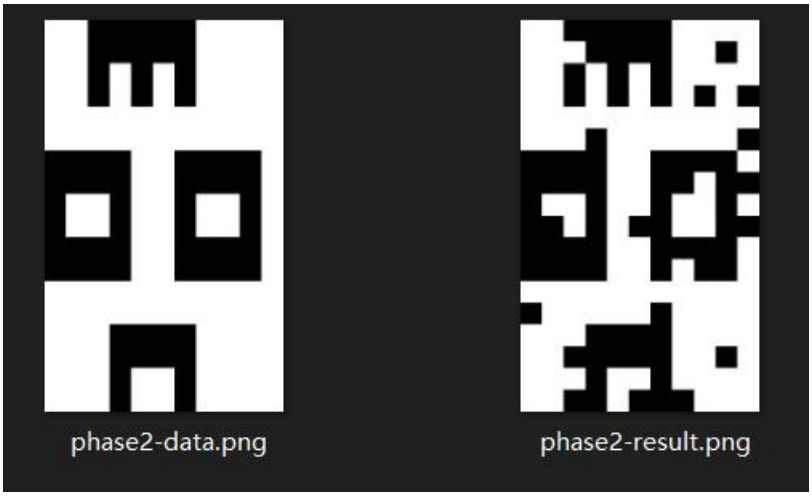
信源数据和解码后的图片没有差异。因为 C5.1 的噪声信道只在每 15 个比特中产生一位错误，汉明码解码时可以纠正一位错误。

3. C5.2——2 比特错误

在 Windows 下运行命令结果：

```
H:\THINGS\study\专业课\信息论与编码\实验\Hamming\codes>lab2_program_windows_amd64.exe -mode=source -phase=2 | python my_
encode.py | lab2_program_windows_amd64.exe -mode=channel -phase=2 | python my_decode.py | lab2_program_windows_amd64.exe
 -mode=verify -phase=2
Phase 2: Verify ok.
Your images: phase2-data.png, phase2-result.png
```

生成的可视化图片：



信源数据和解码后的图片有差异。因为 C5.2 的噪声信道只在每 15 个比特中产生两位错误，而汉明码解码时只能纠正一位错误。因为信道中每 30 个比特，前 15 个产生一位错，后 15 个产生两位错，且前信源中 11 个比特和后 11 个比特相同，所以从 1 开始奇数序号汉明码可以被正确解码。从图中也可见第 1,3,5……行解码时正确的。

4. C5.3——突发错误

在 Windows 下运行命令结果：

```
H:\THINGS\study\专业课\信息论与编码\实验\Hamming\codes>lab2_program_windows_amd64.exe -mode=source -phase=3 | python my_
encode.py | lab2_program_windows_amd64.exe -mode=channel -phase=3 | python my_decode.py | lab2_program_windows_amd64.exe
 -mode=verify -phase=3
Phase 3: Verify ok.
Your images: phase3-data.png, phase3-result.png
```

生成的可视化图片：

信源数据和解码后的图片有差异。因为 C5.3 的噪声信道每 45 个比特中，前 15 个比特有连续的 3 比特出错，中间 15 个比特和后 15 个比特不会出错。而汉明码只能纠正一位错，因此前 15 比特解码时就会出错。每三个连续 15 比特为一组，后 30 比特解码是正确的。

四、OP4——2 比特错误检测

实现了 OP4.2 的（16,11）扩展汉明码。

编码时在 C3 奇偶校验方法基础上添加总校验位，使最后 1 的个数为偶数。解码则根据有没有错误及总校验位给出四种情况的输出。

```python
def OP4encode(d: str):
    D = '0'+'0'+d[0]+'0'+d[1:4]+'0'+d[4:]
    b = 0
    for i in range(15):
        if D[i] == '1':
            b ^= i+1
    p = bin(b)[2:].zfill(4)[::-1]
    r = p[0] + p[1] + d[0] + p[2] + d[1:4] + p[3] + d[4:]
    count = 0
    for x in r:
        if x == '1':
            count += 1
    if count % 2 == 1:
        r = r + '1'
    else:
        r = r + '0'
    return r
```

```python
def OP4decode(r: str):
    b = 0
    for i in range(15):
        if r[i] == '1':
            b ^= i+1
    if b != 0:
        if r[15] == '1':
            print('one error')
            if r[b-1] == '0':
                r = r[0:b-1] + '1' + r[b:15]
            else:
                r = r[0:b-1] + '0' + r[b:15]
        else:
            return 'two errors'
    else:
        if r[15] == '1':
            print('总校验位出错，直接提取数据')
        else:
            print('no error')
    d = r[2]+r[4:7]+r[8:15]
    return d
```

设计测试样例如下图所示，检验了四种可能输出情况，均正确。

```python
def test():
    print('encode test:')
    a = '10100000000'
    print('need: 1011010000000000')
    print('got: ', OP4encode(a), '\n')

    print('decode test:')
    b = '1011010000000000'
    print('need: 10100000000')
    print('got: ', OP4decode(b), '\n')

    print('one error test:')
    c = '1001010000000001'
    print('need: 10100000000')
    print('got: ', OP4decode(c), '\n')

    print('two errors test:')
    d = '1000010000000000'
    print('need: two errors')
    print('got: ', OP4decode(d))
```

```
encode test:
need: 1011010000000000
got:  1011010000000000

decode test:
need: 10100000000
no error
got:  10100000000

one error test:
need: 10100000000
one error
got:  10100000000

two errors test:
need: two errors
got:  two errors
```

五、OP5——突发错误

实现了进行三个分组的比特交织的 (15, 11) 汉明码。交织函数和解码时的恢复函数如下所示。原本想考虑一下如果信源长度不是 33 的倍数情况，经试验发现本测试中无此特殊情况。

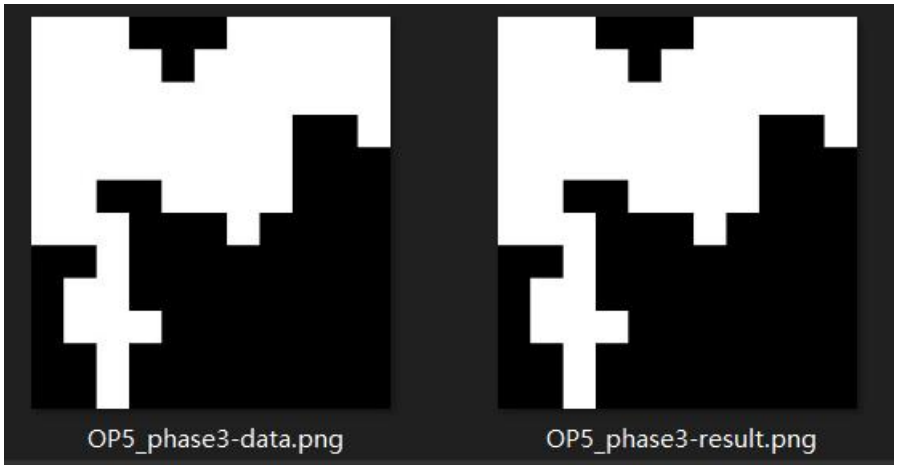
```python
def intertwine(R: list):
    # m = len(R) % 3
    # for i in range(3-m):
    #     R.append('0'*15)
    L = len(R)
    res = []
    for i in range(0, L, 3):
        a, b, c = R[i], R[i+1], R[i+2]
        A, B, C = '', '', ''
        for j in range(0, 15, 3):
            A += a[j]+b[j+1]+c[j+2]
            B += b[j]+c[j+1]+a[j+2]
            C += c[j]+a[j+1]+b[j+2]
        res.append(A)
        res.append(B)
        res.append(C)
    return res
```

```python
def recover(r: str): # 假设编码时分组正好是3的倍数
    R = []
    for i in range(0, len(r), 15):
        R.append(r[i:i+15])
    res = []
    for i in range(0, len(R), 3):
        A, B, C = R[i], R[i+1], R[i+2]
        a, b, c = '', '', ''
        for j in range(0, 15, 3):
            a += A[j]+C[j+1]+B[j+2]
            b += B[j]+A[j+1]+C[j+2]
            c += C[j]+B[j+1]+A[j+2]
        res.append(a)
        res.append(b)
        res.append(c)
    return res
```

重新进行 C5.3 测试，

```
H:\THINGS\study\专业课\信息论与编码\实验\Hamming\codes>lab2_program_windows_amd64.exe -mode=source -phase=3 | python OP5_encode.py | lab2_program_windows_amd64.exe -mode=channel -phase=3 | python OP5_decode.py | lab2_program_windows_amd64.exe -mode=verify -phase=3
Phase 3: Decode cotains error, check your code.
 Got: 000111000000000100000000000000000000000011000000000111001100001110001110111111011111111110011111111110001111111110111111111111111
Need: 001001000000000100000000000000000000011100001100000000011100110000111000100111111101111111110011111111100010000111101111111111111111
Your images: phase3-data.png, phase3-result.png
```

生成图片如下：（为区分 C5.3 中图片，已改名）



两张图片相同。因为 C5.3 的噪声信道是每 45 比特，前 15 比特有连续 3 比特错，后 30 比特无措，因此那三个错就被分到了三组，变成了 C5.1 的情况，可以纠错。