Алгоритм циклического кодирования.

Для удобства построения циклических кодов определим функцию поэлементного сложения по модулю два кодов:

```
def sum_mode_to(data_1, data_2):
    res = []
    for i in range(len(data_1)):
        res.append((data_1[i] + data_2[i]) % 2)
    return res
```

Также разработан интерфейс деления многочленов с бинарными коэффициентами:

```
def div(word, key, n_u):

    #
    print('*--' * 20)
    print('\t'.join(list(word)), ' |__ ', key, ' __')

    for _ in range(n_u):
        d = sum_mode_to(list(map(int, list(word[word.find('1'):word.find('1') + n]))), list(map(int, list(key))))
        if len(d) != len(key):
            break
        s = list(word[:word.find('1')].replace('0', ' ')) + d + list(word[word.find('1') + n:])

        #
        print('\t'.join(list(' ' * (word.find('1')) + key)))

        word = ''.join(map(str, list(s)))

        #
        print('\t'.join(list(word)))

    # word ---> r
    return word.replace(' ', '0'), d
```

Далее, согласно заданию, вычисляем основные параметры циклических кодов:

```
key = '1011'
word = '1101110' #'1011010'
wrd = word
err = 1
n = len(key)
n_u = 4


w, d = div(word, key, n_u)
```

Соответственно, минимальное кодовое расстояние d = 3, для обнаружения и исправления одной ошибки s = 1, число контрольных битов $[m]$ вычислим, исходя из соотношения: $2^m \geq l + 1$, число информационных битов как разность: $(l - m)$, где $l -$ длина кодовой комбинации.

Образующий многочлен: $x^3 + x + 1$.

Далее реализован сам алгоритм декодирования, который производит деление многочленов, проверяет конечные остатки, осуществляет циклический сдвиг многочленов согласно условию. В конце производим сложение кодовой комбинации и остатка:

```
w, d = div(word, key, n_u)

i = 0
word = wrd
d = list(map(int, w))
while sum(list(map(int, w))) > err:
    i += 1
    wrd = wrd[1:] + wrd[0]
    w, _ = div(wrd, key, n_u)
    print('iter -- > ', i)
print('\n\n\t', '\t'.join(list(wrd)))
print('⊕\t')
print('\t', '\t'.join(list(w.replace('0', ' '))))
r = sum_mode_to(list(map(int, wrd)), list(map(int, w)))
print('\t', '---' * (len(r) + 1))
print('\t', '\t'.join(list(map(str, r))))
print('\n|Final ------> ', r[-i:] + r[:-i])
print()
```

Результат работы программы.

Будем вводить кодовые комбинации и программно находить ошибки в них и исправлять.

1. Исходные данные:

```
key = '1011'
word = '1101110'
wrd = word
err = 1
n = len(key)
n_u = 4
```

```
E:\PythonYandex\work\venv\Scripts\python.exe E:/PythonYandex/work/
*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--
1   1   0   1   1   1   0  |__  1011  __
1   0   1   1
0   1   1   0   1   1   0
    1   0   1   1
    0   1   1   0   1   0
        1   0   1   1
        0   1   1   0   0
            1   0   1   1
            0   1   1   1
*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--
1   0   1   1   1   0   1  |__  1011  __
1   0   1   1
0   0   0   0   1   0   1
iter -- >  1
*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--
0   1   1   1   0   1   1  |__  1011  __
    1   0   1   1
    0   1   0   1   1   1
        1   0   1   1
        0   0   0   0   1
iter -- >  2
      0  1   1   1   0   1   1
⊕
                             1
      -----------------------
      0  1   1   1   0   1   0

|Final ------>  [1, 0, 0, 1, 1, 1, 0]


Process finished with exit code 0
```

2. Исходные данные:

```
key = '1011'
word = '1011010'
wrd = word
err = 1
n = len(key)
n_u = 4
```

```
E:\PythonYandex\work\venv\Scripts\python.exe E:/PythonYandex/
*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--
1   0   1   1   0   1   0  |__  1011  __
1   0   1   1
0   0   0   0   0   1   0


    1  0   1   1   0   1   0
⊕
                       1
    ------------------------
    1  0   1   1   0   0   0

|Final ------>  [1, 0, 1, 1, 0, 0, 0]


Process finished with exit code 0
```

3. Исходные данные:

```
key = '1011'
word = '1000110'
wrd = word
err = 1
n = len(key)
n_u = 4
```

```
E:\PythonYandex\work\venv\Scripts\python.exe E:/PythonYandex/
*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--
1   0   0   0   1   1   0  |__  1011  __
1   0   1   1
0   0   1   1   1   1   0
        1   0   1   1
        0   1   0   0   0
            1   0   1   1
            0   0   1   1
*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--
0   0   0   1   1   0   1  |__  1011  __
            1   0   1   1
            0   1   1   0
iter -- >  1
*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--
0   0   1   1   0   1   0  |__  1011  __
        1   0   1   1
        0   1   1   0   0
            1   0   1   1
            0   1   1   1
iter -- >  2
*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--
0   1   1   0   1   0   0  |__  1011  __
    1   0   1   1
    0   1   1   0   0   0
        1   0   1   1
        0   1   1   1   0
            1   0   1   1
            0   1   0   1
iter -- >  3
*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--
1   1   0   1   0   0   0  |__  1011  __
1   0   1   1
0   1   1   0   0   0   0
    1   0   1   1
    0   1   1   1   0   0
        1   0   1   1
        0   1   0   1   0
            1   0   1   1
            0   0   0   1
iter -- >  4


    1   1   0   1   0   0   0
⊕
                            1
    -----------------------
    1   1   0   1   0   0   1

|Final ------>  [1, 0, 0, 1, 1, 1, 0]
```