你来我往之LFSR

产生流密码

以四bit的LFSR为例，

其转移矩阵为 $\begin{bmatrix} a & b & c & d \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$

右乘一个列向量，结果依然是一个列向量

$$\begin{bmatrix} a & b & c & d \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} a \oplus b \oplus c \oplus d \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

新产生的bit等于 $a \oplus b \oplus c \oplus d$，也就等于 $(a+b+c+d)\%2$

所以如果把LFSR现在状态的每一位记作x1、x2 ……， 反馈多项式每一位记作c1、c2，……

新产生的bit就是 $\sum_{i=1}^{n} x_i c_i \pmod 2$

可以直接由矩阵相乘得到。

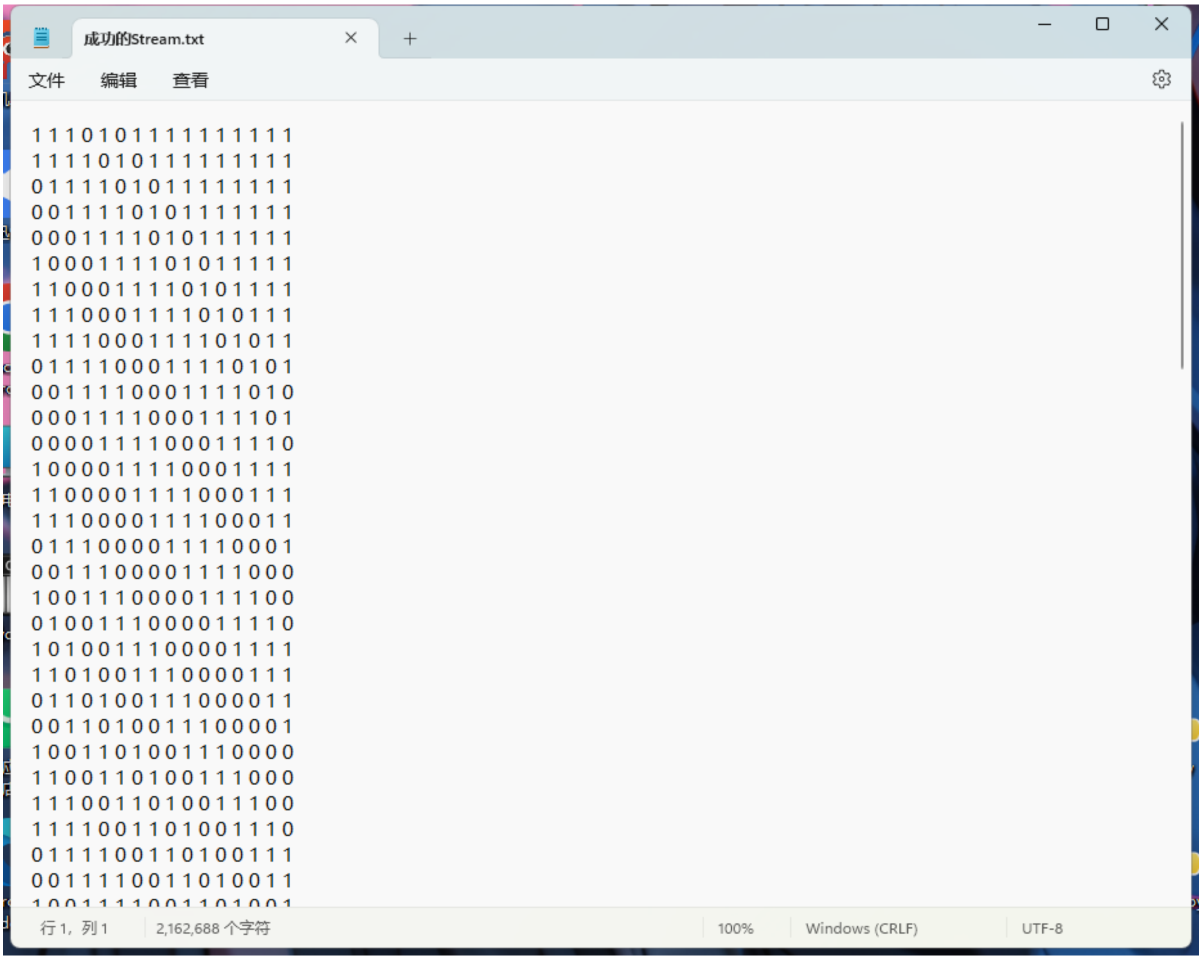
所以，我在python中使用numpy的矩阵和向量完成流密码的产生

```
import numpy as np
..............................
def Create_Stream(matrix, LFSR, count):#产生密码流
    with open("Stream.txt", mode='w') as file:
        for i in range(1, count + 1):
            LFSR = np.dot(matrix, LFSR)
            LFSR[0][0] = LFSR[0][0] % 2
            for row in LFSR:
                for item in row:
                    file.write(str(item)+' ')
            file.write('\n')

#对外暴露出encrypto这个函数作为接口，一键随机生成密码、生成转移矩阵、然后生成流
def encrypto(dimension, count):
    #dimension #寄存器的bit数
    crypto, initial = Create_Crypto_and_Initial(dimension)
    #crypto = [[1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1]]
    print('crypto: ', crypto)
    print('initial: ', initial)
    matrix = Create_Matrix(crypto, dimension)
    print(matrix)
    LFSR = np.array(initial).T
    Create_Stream(matrix=matrix, LFSR=LFSR, count=count)
```

matrix为转移矩阵，LFSR为一个列向量，计算完成后，LFSR的第一位对2取余，然后把寄存器的这一状态记录在“Stream.txt”文本文件中

16bit的长得像这样：



破译流密码

我的思路：

以四bit的为例吧，好叙述一点。

还是设反馈矩阵跟上面的一样，反馈多项式为a b c d

当LFSR密码机出现这样的状态转移时

```
1 0 0 1
1 1 0 0
1 1 1 0
0 1 1 1
1 0 1 1
0 1 0 1
0 0 1 0
1 0 0 1
1 1 0 0
```

从1001到1100，新产生了一个bit 1

可以列出来一个式子，$a \oplus d = 1$

同理从1110到0111，可以列出来一个式子 $a \oplus b \oplus c = 0$

那么这样列上四五个式子（256bit的就列两三百个），一定可以搞出一组解，也就得到了反馈多项式。

但是，这样的方程，是异或运算，不是相加相减，没办法用矩阵直接算。

所以我想到了

例如：

$a \oplus b \oplus d = 1$

$a \oplus b \oplus c = 1$

这样一个异或方程组

写成矩阵的形式$\begin{bmatrix} 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \end{bmatrix}$

对矩阵施展初等行变换，本质上等于两个方程加起来，

$a \oplus b \oplus d \oplus a \oplus b \oplus c = 1 \oplus 1$，等价于

$(a \oplus a) \oplus (b \oplus b) \oplus c \oplus d = 0$

也就是$0 \oplus 0 \oplus c \oplus d = 0$，即$c \oplus d = 0$

这个式子写在矩阵就是$\begin{bmatrix} 0 & 0 & 1 & 1 & 0 \end{bmatrix}$

也就是说

1 1 0 1 1

1 1 1 0 1

这两行加起来得到了

这一行$\begin{bmatrix} 0 & 0 & 1 & 1 & 0 \end{bmatrix}$

这说明，异或方程如果写成矩阵的形式，两行相加，也是模2的

这样的异或方程一样可以写成矩阵的形式，只是两行相加跟普通的矩阵不太一样，

这样的线性方程写成矩阵的形式，两行相加是模2的

这也就意味着无法使用matlab现成的rref功能给增广矩阵消元，解出方程

于是，我自己写了给这种矩阵rref的代码，这也是破译流密码的核心

函数1：用来给两个行向量相加 **（模2）** ，返回这俩加起来的行向量

```
def row_add(row_1, row_2):#功能在于把矩阵的两行加起来（模二），然后返回这一新行
    val = row_1 + row_2
    for i in range(0, len(val)):
        val[i] = val[i] % 2
    return val
```

这里的val row_1 row_2均应该是numpy中的ndarray类型

函数2：交换两行

Python的机制比较特殊，所有变量都是引用，

想要交换两个东西的值

如果直接用t=a，a=b，b=a的形式，会让t跟a其实是引用同一个东西，a变t也变，没法完成交换

所以搞出了此函数，方便一点

```
import copy
def row_exchange(matrix, index_1, index_2):#功能在于交换矩阵的两行
    temp = copy.deepcopy(matrix[index_1])
    matrix[index_1] = matrix[index_2]
    matrix[index_2] = temp
```

函数3：找一个有主元的行

在给矩阵消元时，我的思路是一个主元一个主元来，用主元消去那一列其余的所有1。

for example,

到第二行的时候，发现第二行第二列是个0，

那么为了让矩阵是rref，就得找个第二列是1的行，跟第二行交换

这个函数的作用就是找出一个这样的行的下标，然后用函数2：交换两行，

```
def search_pivot(matrix, start_index, pivot_index):
#从第start_index行开始，找一个有主元（也就是第pivot_index个元素是1）的行，返回其下标
    for index in range(start_index, len(matrix)):
        if matrix[index][pivot_index] == 1:
            return index
    return -1
```

核心函数：rref

```
def rref(matrix):
    row, column = matrix.shape
    size = row if row<column else column
    for i in range(0, size):#一列一列来，先找主元，用主元消掉这一列其他所有1
        if matrix[i][i] == 0:#没有主元,换一个有主元的行上来
            idx = search_pivot(matrix=matrix, start_index=i, pivot_index=i)#从这一行往下的行找主元，有解的情况下，必然能找到主元。
            row_exchange(matrix=matrix, index_1=i, index_2=idx)#这一行与有主元的那一行交换，以构成阶梯型。
        for j in range(0, row):#消掉这一列所有的1
            if not j == i:
                if matrix[j][i] == 1:#如果是1，通过行相加，消掉1
                    matrix[j] = row_add(matrix[j], matrix[i])
        #print('第',i,'次消元结果为')
        #print(matrix)
```

重复一套流程：

- 判断主元是否是0，是0就 找一个这一列是1的行，两行交换，是1就啥也不做
- 遍历其余的所有行，如果发现这一列是1，则把当前行与现在这个主元所在的行相加，消去1

注释掉的两行如果去掉#，可以显示消元每一步的细节

函数5：

这个很简单，就是读txt文件，这个txt中存放了n个寄存器的连续状态

根据这个txt创建出矩阵

然后对矩阵进行消元，把rref型存到名为“result.txt”的txt中

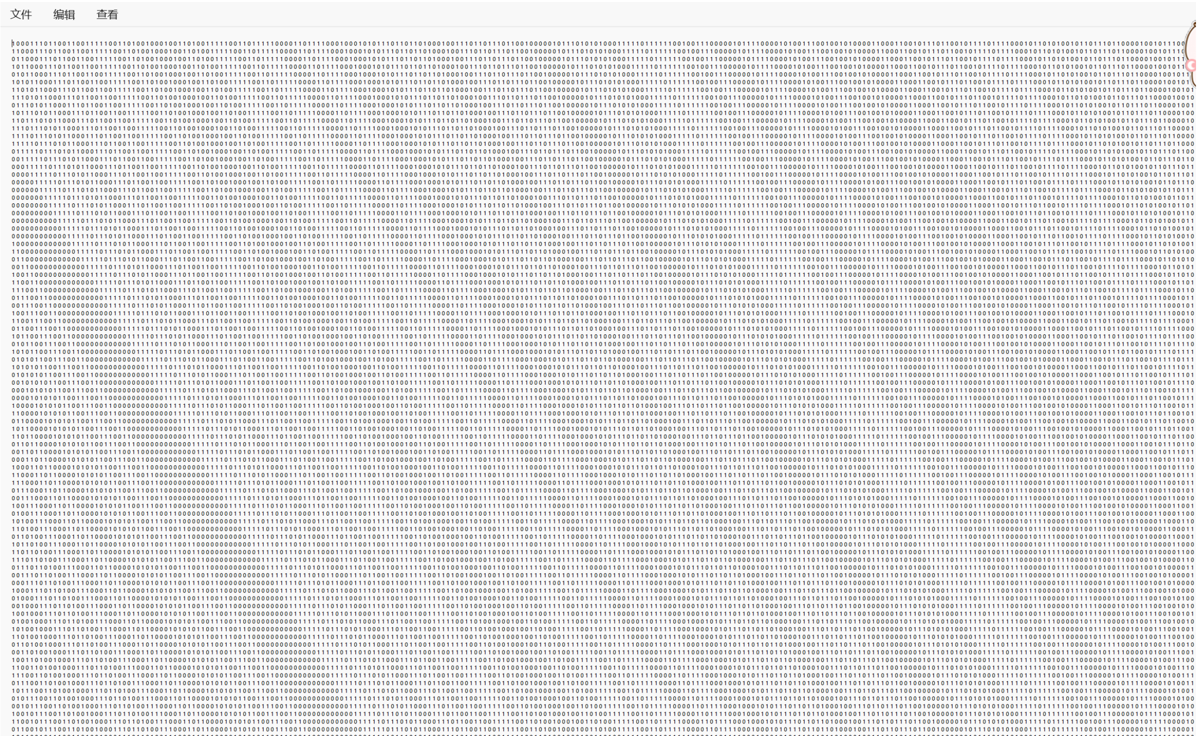
（由于经常会出差错，所以人必须得看一眼矩阵…）

```
def decrypto():#根据同目录下txt文件存储的比特流，进行解密。
    temp = []
    with open('Stream.txt', mode='r') as file:
        for line in file.readlines():
            line = line.split()
            val = [int(i) for i in line]
            temp.append(val)#把字符串形式的玩意全转换成int型
    for i in range(0, len(temp) - 1):
        temp[i].append(temp[i+1][0])#等于是说，把矩阵增了一列
    temp.pop()#然后扔掉最后一行，因为这行没东西可以让他增加一列
    matrix = np.array(temp)
    rref(matrix)
    with open('result.txt', mode='w') as rst:
        for row in matrix:
            for item in row:
```

```
                rst.write(str(item)+' ')
            rst.write(' \n')
```

256bit的破译实例

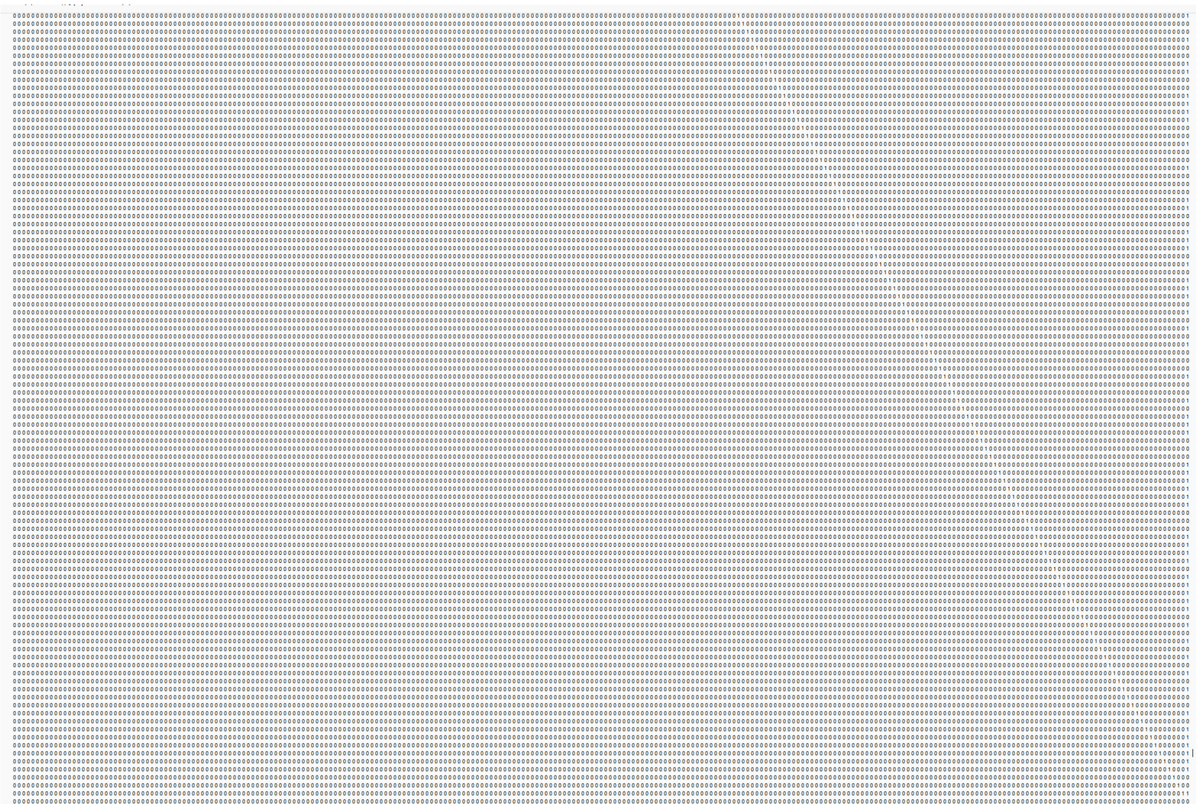
同学给过来的状态



（一大堆0 1，随作业附上此txt）

（可以根据此图片的纹路明显感觉出移位……）

处理之后，rref矩阵长这样：



图中很明显的一条斜线就是主元

（txt随作业附上）

同学给的密码是

crypto: [[1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1]]

与txt的最后一列（除掉全零行）完全重合

完美破译密码

--至于LFSR的初始值，这个很好办，反推一下就可以了。

把第一个状态左移一位。设最后一位是x，根据：

- 第二个状态序列的第一位
- 带有未知数x的初始序列
- 反馈多项式

三者列一个式子，很容易解出来

initial: [[1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1]]

一些残留问题

rref的时间复杂度太大

假设矩阵m行n列，出现1的概率是0.5

那么光两行相加这个操作就要执行(mn)/2次，

此外还有行交换、、等等，当样本大起来时，电脑算的很慢

有时，rref矩阵并不是单位阵+列向量，也就是说一套状态转移，可能会存在多套密码

这个现象很奇怪，以6bit数说明吧

反馈多项式是这样的情况下：crypto：[[0, 0, 1, 0, 1, 0]]

产生了这样10个状态转移

1 0 1 1 1 0
0 1 0 1 1 1
1 0 1 0 1 1
0 1 0 1 0 1
0 0 1 0 1 0
0 0 0 1 0 1
0 0 0 0 1 0
1 0 0 0 0 1
0 1 0 0 0 0
0 0 1 0 0 0

解出来的rref矩阵是这样的
1 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 1 0
0 0 0 0 1 0 1
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0

可以看到，第六行没有主元，第六个未知数在第一行和第四行出现

这就是说第六个未知数可以是0，也可以是1，有两套解法

密码可以是[0,0,1,0,1,0]**(当然这是本来的密码)**,也可以是[1,0,1,1,1,1]

如果密码是[1,0,1,1,1,1]，会发现仍然符合这一套状态转移

例如1 0 1 1 1 0 的1、3、4、5、6位一起异或起来就是0，下一个状态0 1 0 1 1 1

0 1 0 1 1 1 的1、3、4、5、6位一起异或起来就是1，下一个状态 1 0 1 0 1 1

这两种多项式都是对的。

那，如果两行没有主元的时候，理论上就有4套密码。

我起初以为这**仅仅**是样本数量不够导致的，

因为很显然，监听到的状态数太少，有些方程可能是等价的（线性相关了）

n个bit最大周期是2^n

但是。对6bit的LFSR，取64个状态，

出现这样的状态转移时

（很明显状态转移早已出现了循环）

1 0 0 1 1 0
1 1 0 0 1 1
0 1 1 0 0 1
0 0 1 1 0 0
1 0 0 1 1 0
1 1 0 0 1 1
0 1 1 0 0 1
0 0 1 1 0 0
1 0 0 1 1 0
1 1 0 0 1 1
0 1 1 0 0 1
0 0 1 1 0 0
1 0 0 1 1 0
1 1 0 0 1 1
0 1 1 0 0 1
0 0 1 1 0 0
1 0 0 1 1 0
1 1 0 0 1 1
0 1 1 0 0 1
0 0 1 1 0 0
1 0 0 1 1 0
1 1 0 0 1 1
0 1 1 0 0 1
0 0 1 1 0 0
1 0 0 1 1 0
1 1 0 0 1 1
0 1 1 0 0 1
0 0 1 1 0 0
1 0 0 1 1 0

1 1 0 0 1 1
0 1 1 0 0 1
0 0 1 1 0 0
1 0 0 1 1 0
1 1 0 0 1 1
0 1 1 0 0 1
0 0 1 1 0 0
1 0 0 1 1 0
1 1 0 0 1 1
0 1 1 0 0 1
0 0 1 1 0 0
1 0 0 1 1 0
1 1 0 0 1 1
0 1 1 0 0 1
0 0 1 1 0 0
1 0 0 1 1 0
1 1 0 0 1 1
0 1 1 0 0 1
0 0 1 1 0 0
1 0 0 1 1 0
1 1 0 0 1 1
0 1 1 0 0 1
0 0 1 1 0 0
1 0 0 1 1 0
1 1 0 0 1 1
0 1 1 0 0 1
0 0 1 1 0 0
1 0 0 1 1 0
1 1 0 0 1 1
0 1 1 0 0 1
0 0 1 1 0 0
1 0 0 1 1 0
1 1 0 0 1 1
0 1 1 0 0 1
0 0 1 1 0 0
1 0 0 1 1 0

矩阵是这样的：

1 0 0 1 1 0 1
0 1 0 1 0 1 1
0 0 1 1 0 0 1
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0

缺失2个主元。

这也就是说明密码和初值随机的不好的情况下，LFSR就是可能会有多套适合的密码吗？