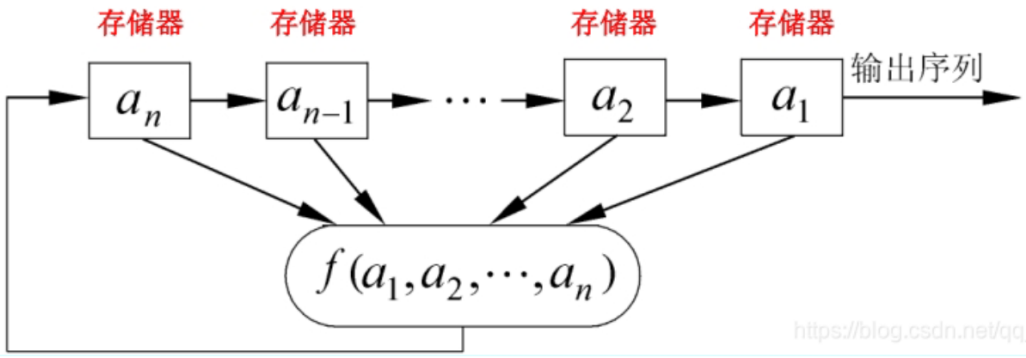
LFSR

概念了解

LFSR是指线性反馈移位寄存器，给定一定的输出，将该输出的线性代数再用作输入的移位寄存器，异或运算是最常见的单比特线性代数： 对寄存器的某些位进行异或操作后作为输入,再对寄存器中的各比特进行整体移位.该结构具有结构简单,运行速度快的特点,常被应用于伪随机数和伪随机噪声的生成中.同时,该原件常与流密码相关部分联合使用.

原理

LFSR是FSR的一种，还有一种是NFSR(非线性)



函数可表示为:

$$f(a_1,a_2,\cdots,a_n)=c_1a_n \oplus c_2a_{n-1} \oplus \cdots \oplus c_na_1$$

这里给不了解布尔运算的做一个基础补充： 布尔运算分为与运算，或运算，异或运算和非运算

与运算:

1. 与（AND）

- **逻辑表达**：A AND B
- **结果**：只有当 A 和 B 都为1时，结果才为1（真）。
- **真值表**：

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

或运算

2. 或（OR）

- **逻辑表达**：A OR B
- **结果**：只要 A 或 B 至少有一个为1，结果为1。
- **真值表**：

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

异或运算

3. 异或（XOR）

- **逻辑表达**：A XOR B
- **结果**：当 A 和 B 中恰好有一个为1时，结果为1。
- **真值表**：

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

非运算

4. 非（NOT）

- **逻辑表达**：NOT A
- **结果**：如果 A 为1，结果为0；如果 A 为0，结果为1。
- **真值表**：

A	NOT A
0	1
1	0

- an+1=c1an⊕c2an-1⊕...⊕cna1
- an+2=c1an+1⊕c2an⊕...⊕cna2
- ...
- an+i=c1an+i-1⊕c2an+i-2⊕...⊕cnai(i=1,2,3...)

例如:

反馈函数为：$a5 + i = a3 + i \oplus a_i$，(i = 1,2,...)可以得到输出序列为：

10011010010000101011101100011111 100110...

周期为31。

题型:1.已知反馈函数，输出序列，求逆推出初始状态

```
from flag import flag
assert flag.startswith("flag{")
assert flag.endswith("}")
# 作用：判断字符串是否以指定字符 开头或结尾
assert len(flag)==25

def lfsr(R,mask):
    output = (R << 1) & 0xffffff #将R向左移动1位,
bin(0xffffff)='0b111111111111111111111111'
    i=(R&mask)&0xffffff          #按位与运算符&：参与运算的两个值,如果两个相应位都为1,
则该位的结果为1,否则为0
    lastbit=0
    while i!=0:
```

```
        lastbit^=(i&1)           #按位异或运算，得到输出序列
        i=i>>1
    output^=lastbit              #将输出值写入 output的后面
    return (output,lastbit)

R=int(flag[5:-1],2)  #flag为二进制数据
mask    =   0b10100110001000110 0

f=open("key","ab")   #以二进制追加模式打开
for i in range(12):
    tmp=0
    for j in range(8):
        (R,out)=lfsr(R,mask)
        tmp=(tmp << 1)^out
    f.write(chr(tmp))   #将lfsr输出的序列每8个二进制为一组，转化为字符，共12组
f.close()
```

思路:

题目已知条件为 flag长度为19bits,mask长度也为19bits.

由LFSR的输出序列{ a_n }满足的条件:

- $an+i=c_1\,an+i-1\oplus c_2\,a_{n+i-2}\oplus\ldots\oplus c_n\,a_i$ (i = 1,2,3,...)

可知，输出值$an+i$的结果与c的值相关，即题目中的mask。只有当c的值为1时，$c_1\,an+i-1,\ldots,c_n\,a_i$ 的值才可能为1

题目中mask中只有第（3，4，5，9，13，14，17，19）位为1，其余都是0(mask这里右边才是第一位，从右往左增大)

现在我们的目的就是为了求出前19位seed的值，而我们已知了seed后面输出序列的值（题目中给的附件key.txt）。那么我们逆推就能得到seed的值了。lfsr(R,mask)函数执行的是19bits的值。那么我们获取到输出序列前19bits值，即:

key = 0101010100111000111

现在需要计算$a19$的值，假设我们将 R = $a1901010101001110001 1$,进行lfsr(R,mask)运算，那么我们将得到输出值为 key[-1]=1。

因为mask中只有第（3，4，5，9，13，14，17，19）位为1，所以线性反馈函数只取这几位对应的a值

1=a_{19} (R[-3])(R[-4])(R[-5])(R[-9])(R[-13])(R[-14])^(R[-17])

得1=$a19$^0,得到$a19$=1

同理：R = $a18a1901010101001110001$ 的输出值为 key[-2]=1，求得a_{18} =1

第一种方法:

```
from Crypto.Util.number import*

f = open('key.txt','rb').read()
r = bytes_to_long(f)
bin_out = bin(r)[2:].zfill(12*8)
R = bin_out[:19]    #获取输出序列中与掩码msk长度相同的值
print(R)
mask = '1010011000100011100'  #顺序 c_n,c_n-1,。。。,c_1
key =  '0101010100111000111'

R = ''
```

```
for i in range(19):
    output = 'x'+key[:18]
    out =
int(key[-1])^int(output[-3])^int(output[-4])^int(output[-5])^int(output[-9])^int(o
utput[-13])^int(output[-14])^int(output[-17])
    R += str(out)
    key = str(out)+key[:18]

print('flag{'+R[::-1]+'}')
```

第二种方法：猜seed

```
from Crypto.Util.number import*
import os,sys
os.chdir(sys.path[0])

f = open('key.txt','rb').read()
c = bytes_to_long(f)
bin_out = bin(c)[2:].zfill(12*8)   #将key文本内容转换为 2 进制数，每个字节占 8 位

R = bin_out[0:19]  #取输出序列的前19位
mask = 0b10100011000100011100

def lfsr(R,mask):
    output = (R << 1) & 0xffffffff
    i=(R&mask)&0xffffffff
    lastbit=0
    while i!=0:
        lastbit^=(i&1)
        i=i>>1
    output^=lastbit
    return (output,lastbit)

#根据生成规则，初始状态最后一位拼接输出序列
#我们可以猜测seed的第19位（0或1），如果seed19+R[:18]输出值等于R[:19]，那么就可以确定
seed值了
def decry():
    cur = bin_out[0:19]      #前19位 2 进制数
    res = ''
    for i in range(19):
        if lfsr(int('0'+cur[0:18],2),mask)[0] == int(cur,2):
            res += '0'
            cur = '0'+cur[0:18]
        else:
            res += '1'
            cur = '1' + cur[0:18]
    return int(res[::-1],2)

r = decry()
print(bin(r))
```

第三种方法:

```
import os,sys
os.chdir(sys.path[0])
from Crypto.Util.number import *
key = '0101010100111000111'
mask = 0b10100011000100011100

R = ""
index = 0
key = key[18] + key[:19]
while index < 19:
    tmp = 0
    for i in range(19):
        if mask >> i & 1:
            tmp ^= int(key[18 - i])
    R += str(tmp)
    index += 1
    key = key[18] + str(tmp) + key[1:18]

print (R[::-1])
```