#接收两个参数，R是32位的初始状态(即flag)，mask是32位的掩码，由于mask已知，所以我们就直接把他当做一个常数即可。

def lfsr(R,mask):

#把R左移一位后低32位（即抹去R的最高位，然后在R的最低位补0）的值赋给output变量。

output = (R << 1) & 0xffffffff

#把传入的R和mask做按位与运算，运算结果取低32位，将该值赋给i变量。

i=(R&mask)&0xffffffff

#从i的最低位向i的最高位依次做异或运算，将运算结果赋给lastbit变量。

lastbit=0

while i!=0:

lastbit^=(i&1)

i=i>>1

#将output变量的最后一位设置成lastbit变量的值。

output^=lastbit

#返回output变量和lastbit变量的值，output即经过一轮lfsr之后的新序列，lastbit即经过一轮lfsr之后输出的一位。

return (output,lastbit)

通过上面的分析，我们可以看出在这道题的情境下，lfsr函数本质上就是一个输入R输出lastbit的函数，虽然我们现在已经清楚了R是如何经过一系列运算得到lastbit的，但是我们前面的反馈函数都是数学表达式的形式，我们能否将上述过程整理成一个表达式的形式呢？这就需要我们再进一步进行分析：

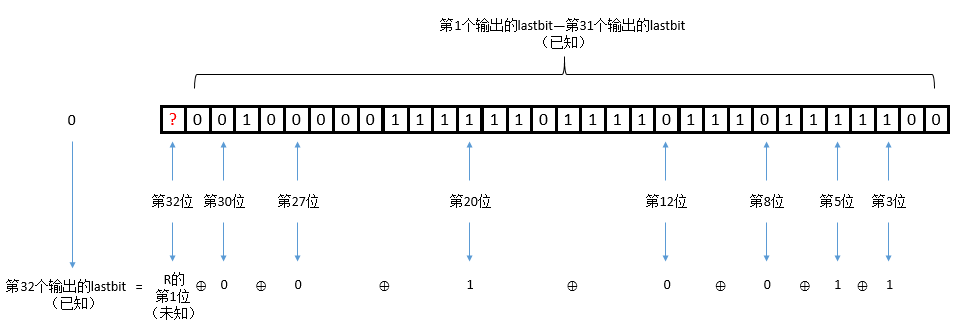
* mask只有第3、5、8、12、20、27、30、32这几位为1，其余位均为0。
* mask与R做按位与运算得到i，当且仅当R的第3、5、8、12、20、27、30、32这几位中也出现1时，i中才可能出现1，否则i中将全为0。
* lastbit是由i的最低位向i的最高位依次做异或运算得到的，在这个过程中，所有为0的位我们可以忽略不计（因为0异或任何数等于任何数本身，不影响最后运算结果），因此lastbit的值最后是0还是1仅取决于i中有多少个1：当i中有奇数个1时，lastbit等于1；当i中有偶数个1时，lastbit等于0。
* 当R的第3、5、8、12、20、27、30、32这几位依次异或结果为1时，即R中有奇数个1，因此将导致i中有奇数个1；当R的第3、5、8、12、20、27、30、32这几位依次异或结果为0时，即R中有偶数个1，因此将导致i中有偶数个1。
* 因此我们可以建立出联系：lastbit等于R的第3、5、8、12、20、27、30、32这几位依次异或的结果。

将其写成数学表示式的形式，即为：

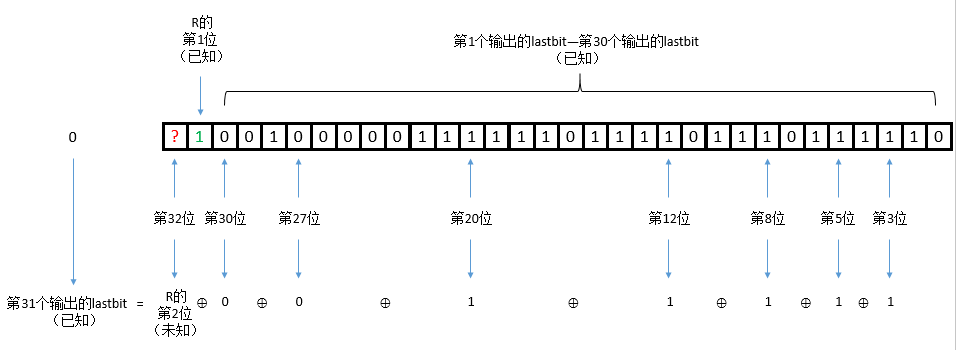


显然，lastbit和R之间满足线性关系，那么接下来我们就可以开始求解了：

我们想象这样一个场景，当即将输出第32位lastbit时，此时R已经左移了31位，根据上面的数学表达式，我们有：



这样我们就可以求出R的第1位，同样的方法，我们可以求出R的第2位：



以此类推，R的全部32位我们都可以依次求出了。