[用位运算巧解元素出现次数问题](http://blog.csdn.net/dinosoft/article/details/6443354)

先举道简单的题：

有一些筷子，长短不一，长度一样的能组成一双。现在刚好有一堆筷子，数量为奇数，而且刚好只有一只筷子落单，其余都成双。

请找出这只落单的筷子的长度。

在acm的书看到有类似的题目，是用位运算解的，挺巧妙的。

有几种办法：

1. 先排序，再扫描一次。如果扫描到长度相同的连续一段个数为奇数，则输出解。例如1，1，2，2，2，3。。。 2出现的次数为3，奇数，输出2，可以停了。O(NlogN)  (用于排序）
2. 筷子长度的范围比较小的话，可以开个数组。a[i] 记录长度为i的筷子出现的次数，累加后,a[i]=a[i]%2, 最后扫描一遍，找出a[i]为一的那个下标i。O(N)

而书里提供了一个巧妙的办法，对每个长度x，求ans ^=x

^是xor，异或运算，可以理解成 对x中的位为1的取反。比如数字 1111 1111（2） 如果想翻转后面四位，可以这样 1111 1111 ^ 0000 1111=1111 0000

好了，现在只要一双筷子出现xor两次之后，位翻转就抵消掉了，所以～ans最后保存出现一次的筷子长度.

今天群里有人发了另外一道题

是易总状态里的一道题目，状态回复写程序会格式错乱，于是发篇日志贴一下程序。

题目如下：一个全是32位整数的大数组，除了其中一个数字出现2次外，其余的数字都出现了3次。如何找出那个只出现了两次的数字？

这道题我以前看过，那个版本是其中一个数字出现1次，不过都是一样的，不影响算法。当时看到解答的程序感觉非常惊艳，为其简洁与精妙赞叹了许久，以下为程序部分：

（**相当于构造一个3进制的xor, 平时的xor相当于对和mod 2，1+1=0 0+1=1 0+0=0  
对和mod 3的话，1+1=2 1+2=0 2+2=1**）

**[python]** [view plaincopy](http://blog.csdn.net/Dinosoft/article/details/6443354)

1. **def** special(lst):
2. ones = 0
3. twos = 0
4. **for** x **in** lst:
5. twos |= ones & x
6. ones ^= x
7. not\_threes = ~( ones & twos )
8. ones &= not\_threes
9. twos &= not\_threes
10. **return** ones

其中ones记录了所有出现了(模3余1)次的bit，twos记录的是余2的。not\_threes是当一个bit出现第3次的时候，把它清掉。

我首先想到的也是位运算。不过这道题出现次数改了，处理起来会麻烦点，但思路还是一样的。

要点：

1.  一个数字每出现3次，我们就要把它消掉。拓展一下，我们只要对每个位置的1每出现3次就消掉。（数据保证其他数字出现3次，所以才能这么做）  
比如：  
100  
101  
100  
第一位（左起）的1出现3次，第二位0次，第三位1次。消掉后为001

2.  x%3 可以为0，1，2。上面的步骤去掉了0的情况，所以我们还要保存1，2的情况。

然后我们回过头来理解那个程序

先看  ones ^= x，如果循环里面只有这一句的话，ones求的就是取反1次，3次，5次。。。，因为每满3次要清掉，所以循环就变成1，1，1。。了。

**[python]** [view plaincopy](http://blog.csdn.net/Dinosoft/article/details/6443354)

1. not\_threes = ~( ones & twos )
2. ones &= not\_threes
3. twos &= not\_threes

而这一块就是每满3就清除的代码。注意这一块 ( ones & twos )，ones求的是余1，twos求的是余2，原本交集是空的，但这里ones还没过滤3的情况，而开头 twos |= ones & x 求的其实是之前2（现在可能变3了）或者本轮确定为2的情况。所以 ( ones & twos )取到3的情况(即ones的3和twos变成3的部分）

瞎了。。这个程序真TM难啊。。。

附：我自己调试的代码

**[python]** [view plaincopy](http://blog.csdn.net/Dinosoft/article/details/6443354)

1. #!/usr/bin/env python
2. **import** random
3. **def** special(lst):
4. ones = 0
5. twos = 0
6. **for** x **in** lst:
7. twos |= ones & x
8. ones ^= x
9. not\_threes = ~( ones & twos )
10. **print** "%x %x %x/n" %( ones,twos,~not\_threes)
11. ones &= not\_threes
12. twos &= not\_threes
13. **return** ones
14. **if** \_\_name\_\_=="\_\_main\_\_":
15. ll=[4,5,4]
16. **print** special(ll)

题目：有n个数，其中绝大部分数出现了正好3次，只有一个数只出现了1次，请写一段程序把这个数找出来

解法1： 建一个hash table，把number作为key，遍历数组一遍，对出现过的数进行计数，最后再遍历一遍找到只出现一次的数。

空间复杂度:length(为了时间上高效，一般length要取到Theta(n))

时间复杂度： n^2/length+0.5n（n时间遍历一遍，每次要查找n/length的hash table,最后再遍历一遍）

解法2：对数组进行select，对小于pivot与大于pivot的数分别计数，对不能被3整除的一边进行递归调用。

 空间复杂度：如果不在乎源数据被改变是可以仅用O(1)的空间的，但通常来讲需要O(n)空间来备份

 时间复杂度：O(n)

解法3（感谢雪瑶姐提供）:

public static int findSpecial(int[] array){  
int ones = 0;  //出现1次的1  
**int twos = 0; //出现2次的1  
int not\_threes;  //出现3次的1取反**

**for(int i :array){  
twos |= ones & i;   
ones ^= i;  
not\_threes = ~(ones&twos);  
ones &= not\_threes;  
twos &= not\_threes;  
}  
return ones;  
}  
}**  
 空间复杂度O(1),时间复杂度也是O(n),不会改变原数组，应该是最好的算法了

 其中ones记录了所有出现了模3余1次的bit，twos记录的是余2的。not\_threes是当一个bit出现第3次的时候，把它清掉。

最后输出ones（如果题目中那个特殊的数出现了1次，当然如果是出现2次的话，应该输出twos。）

但是，这道题的讨论还没有完。假如把题目改成“有n个数，其中绝大部分数出现了正好2次，只有一个数出现了1次，请把这个数找出来” 有一个只需遍历一遍且不用额外空间的算法：把所有数字异或一遍，最后结果就是只出现1次的那个数。 那么，可能有人会想，在这道题的条件下，是不是也存在一个类似于异或的运算符#，可以仅把所有数#一遍以后就得到结果呢？、

答案是否定的。假如存在这个运算符#，那它必须满足下面这些条件：

 A={所有32位2进制数}  #:A->A  
交换律 a#b=b#a  
结合律 a#b#c=a#(b#c)  
存在e 对于任意a a#e=a  
对于任意a a#a#a=e

我们可以证明满足上面条件的运算符不存在：

首先我们定义一个负元-a：a#-a=e  
由a#a#a=e 可知-a存在,根据结合律，-a=a#a (1)  
假设-a不唯一,-a=b或c, 那么a#b#c=(a#b)#c=c=(a#c)#b=b（交换律&结合律）矛盾 (2)  
由（1）（2）可知，-a存在且唯一。显然，-e=e  
由交换律可知，如果a#b=e=>b#a=e，所以负元这个relation是可逆的, -a=b => -b=a  
因为-e=e并且A是一个**有限偶数项**集合  
所以必定还有一个数x，-x=x  
那么x#x#x=e#x=x与题设里x#x#x=e矛盾

综上所述，这个运算符并不存在

感谢UTD同学在证明上的帮助