**191. Number of 1 Bits**

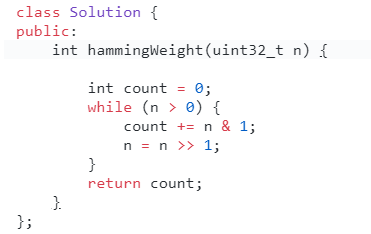
C++跟Java的位运算相同！

注意几个位运算符号：相与&，相或|，异或^

都是按每个位进行运算的。

注意**Java不支持无符号类型!**

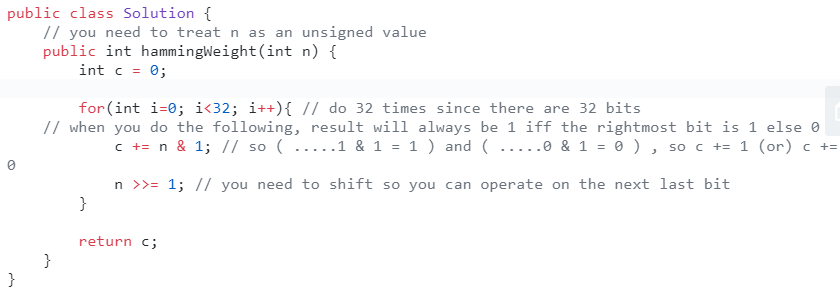
在此题中，C++可以这样写：



While循环内是n > 0，因为参数是无符号32位类型整数。

而Java并不是，所以判断条件不能是n > 0。

所以Java中的解法是：



32位，所以循环32次就可以了。

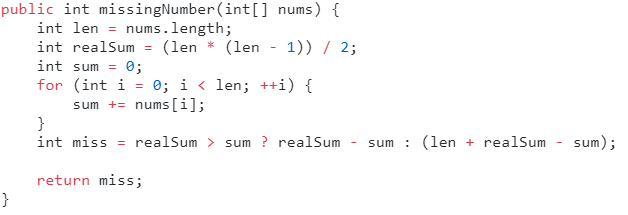
同理，C++中也可以这么写，因为C++的参数也是32位整数。

当然也可以先判断n是否是正数，但比较麻烦了。

**268. Missing Number**

规定为线性复杂度。

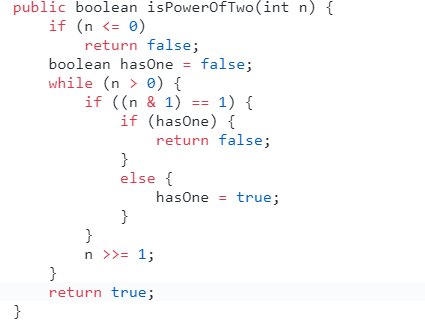
自己注意到数组的特点，可以根据和运算求得差值：



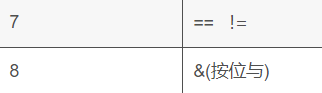
**231. Power of Two**

题目分析: 2的整数次幂对应的二进制数只含有0个或者1个1，所以我们要做的就是判断输入的数的二进制表 达形式里是否符合这一条件。 有一种corner case需要注意，当输入的数为负数的时候，一定不是2的幂。

当然还有个问题，就是表达式if (n & 1 == 1)一直报错“ bad operand types for binary operator '&'”，因为**相对于位运算符，恒等运算符优先级大一点，这样就导致了“&”左边是int型，右边是boolean型(基础)。**



注意：这是Java中的运算符优先级，== > &，C++则相反？：

——Java

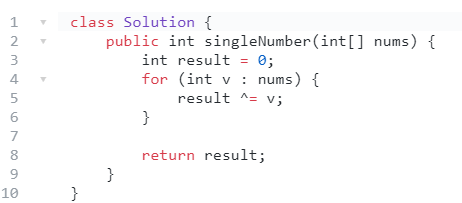
——C++

有点奇怪，是不是没更新？

**136. Single Number**

**一个数组中一个数出现一次，其他数都出现两次，找出这个只出现一次的数。**

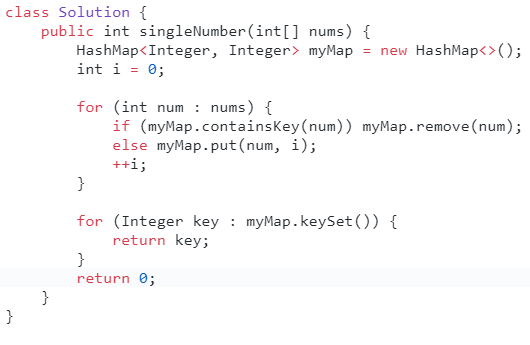
根据密码学知识(?)，x ^ y ^ x = y，因为异或的作用是相同的数异或为0，不同的不为0。所以只要遍历数组异或即可。



这有个小问题，为什么结果result从0开始而不是从数组的第一个元素开始，因为0与任何数异或都是数本身(0还是0，1还是1)。

扩展：如果一次数出现一次，其他数出现偶数次，都可以用异或方法找出。

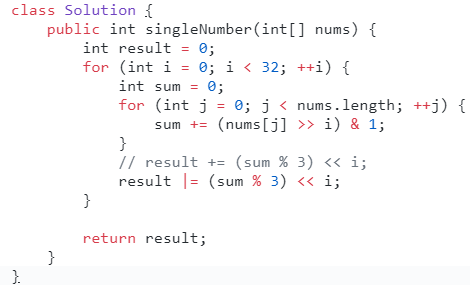
Tip：如果使用HashMap，虽然空间复杂度不是常数级，但有些注意的地方，比如如何提取HashMap的数据：



**137. Single Number II**

**一个数组中一个数出现一次，其他数都出现三次，找出这个只出现一次的数。**

对每一位进行余3处理，而后加进结果即可。



注意要先右移，取余后再左移。

**260. Single Number III**

**一个数组中有两个数出现一次，其他都出现两次，找出这两个只出现了一次的数。**

思路是把数组分成两部分，两个部分分别包含一个只出现了一次的数，然后对每个部分进行遍历异或，就可以找出这两个数。承接了Single Number I的思想。

问题是如何把数组分开判断呢，首先对整个数组遍历异或，得到的数将会是那两个不相同的数异或的结果，我们要对这个结果作为一个判断条件，接下来就是找到这个结果最右位为1的数，对它的相反数相与，得到的就是最右位为1的数，因为最右位为1表示这位的右边都是0，相反数是取补码，则取反加一，相反数的最右位是1，后为0，跟数本身相同，而这个最右位以前的位数都是与相反数相反的。然后以此作为判断条件if语句对数组进行分隔，分别遍历两部分异或即可。

