1. 位运算篇

**位运算基础知识：**

[**http://www.cnblogs.com/zhangziqiu/archive/2011/03/30/2000333.html**](http://www.cnblogs.com/zhangziqiu/archive/2011/03/30/2000333.html)

**LeetCode中关于位运算的题目有以下几道：**

**1.** [**Single Number**](http://blog.csdn.net/linhuanmars/article/details/22648829)

**2.** [**Single Number II**](http://blog.csdn.net/linhuanmars/article/details/22645599)

**3.** [**Divide Two Integers**](http://blog.csdn.net/linhuanmars/article/details/20024907)

**4.** [**Pow(x, n)**](http://blog.csdn.net/linhuanmars/article/details/20092829)

**1. Single Number，题目本身要求是找出唯一一个在数组中出现一次的整数，而其他都会出现两次。这里利用到了位运算中异或的性质，就是两个相同的数进行异或会得到0，并且任何一个数与0的异或还是原数。利用上面的性质，只要把数组中的元素一一异或起来，因为出现两次的会互相抵消，最后会只剩下那个出现一次的整数。这个方法只需要一次扫描，即O(n)的时间复杂度，而空间上也不需要任何额外变量，即O(1)的空间复杂度。**

**代码如下：**

public int singleNumber(int[] A) {

if(A == null || A.length == 0){

return 0;

}

int result = A[0];

for(int i = 1; i < A.length; i++){

result ^= A[i];

}

return result;

}

**2. Single Number** [**II**](http://blog.csdn.net/linhuanmars/article/details/22645599)**，上面的方法就没办法了，因为出现三次就不能利用异或的性质了，所以这个题目得使用另外的方法了。 算法是对每个位出现1的次数进行统计，因为其他元素都会出现三次，所以最终这些位上的1的个数会是3的倍数。如果我们把统计结果的每一位进行取余3，剩下的结果就会剩下那个出现一次的元素。这个方法对于出现k次都是通用的，包括上面的**[**Single Number**](http://blog.csdn.net/linhuanmars/article/details/22648829)**也可以用这种方法，不过没有纯位运算的方法高大上哈。**