leetcode 之 Single Number II

2016年01月28日 00:20:22 yutianzuijin 阅读数：3746 标签： single number 真值表 最小项 异或 xor 更多

问题来源：Single Number II

问题描述：给定一个整数数组，除了一个整数出现一次之外，其余的每一个整数均出现三次，请找出这个出现一次的整数。

大家可能很熟悉另一个题目(Single Number)：除了一个数出现一次之外，其余的均出现两次，找到出现一次的数。该问题很简单，大家肯定都知道解法：将所有的数异或，最后的结果即是出一次的数。用到的知识是A^A=0，两个相同的数异或变为0，然后0^B=B，这样即可找到出现一次的数。

新问题有了变化，出现两次变成出现三次，整个问题的解法就不一样了。如何做到时间复杂度O(n)，空间复杂度O(1)保持不变呢？最笨的方法就是计数，将每一个整数都看成一个长度位32的数组，然后统计32位中每一位出现1的次数。如果一个数出现3次，则其出现1的位肯定也是3次，这时如果某位出现4次，则意味着出现一次的数在该位也为1。通过分析所有的位，我们即可以找到这个出现一次的数。代码如下：

```
int singleNumberII(int* A,int len)
{
        int count[32],result=0;
        memset(count,0,sizeof(int)*32);

        for (int i=0;i<len;i++)
        {
                for (int j=0;j<32;j++)
                {
                        count[j]+=A[i]>>j&0x1;
                }
        }

        for (int i=0;i<32;i++)
        {
                result|=(count[i]%3<<i);
        }

        return result;
}
```

很多人对上面的代码会有一个疑问：上面的代码分配了一个长度为32的数组，这样空间复杂度还算是O(1)吗？答案是肯定的，只要分配的空间是已知的固定值，空间复杂度都是O(1)。另一个例子，统计每个char字符出现的次数分配的长度为128的空间也属于O(1)。

上面的方法可以解决问题，但是显得不够优雅，是否存在一个和原始问题一样优雅的解法呢？答案是肯定的，但是理解起来会比较困难。今天我们就深入剖析一下这种优雅的解法，等你掌握之后就可以很容易地解决一系列的问题。

新解法用到了大学阶段大家都学过的数字逻辑电路知识，莫慌，用到的知识非常浅显，很容易就回忆起来。第一个概念真值表(truth table)，是用0和1表示输入和输出之间全部关系的表格，异或的真值表如下：

A	B	P
0	0	0
0	1	1
1	0	1
1	1	0

其中，A和B是输入，共有四种组合，P是输出。给定一个真值表，我们还需要将其逻辑函数表达式写出来。共有两种方法，最小项推导法和最大项推导法。第二个概念最小项推导法(两个概念都很简单，我们只关注最小项)，把输出为1的输入组合写成乘积项的形式，其中取值为1 的输入用原变量表示，取值为0的输入用反变量表示，然后把这些乘积项加起来。例如，上面异或真值表的逻辑函数表达式可以写为：

$$P = A \cdot \bar{B} + \bar{A} \cdot B = A \oplus B$$

是不是很简单。有了这两个概念，我们就可以介绍新解法了。

一个32位int型整数可以看成32个独立的位，每一位都可以独立考虑，所以后面的描述都单指一个位。当一个数最多出现两次时，我们可以只用1 bit来描述，但是当一个数最多出现三次时，我们必须要用2 bit来描述。针对该问题，可以用00表示一个数未出现，01表示一个数出现一次，10表示一个数出现两次，当出现三次的时候按理应该是11，但是我们将其重置为00表示该数已经达到上限，肯定不是要找的数可以丢掉。所以给定一个数，其出现次数变化规律为00→01→10→00。针对这个变化规律，我们可以得到一个真值表：

high_bit	low_bit	input	high_bit_output	low_bit_output
0	0	0	0	0
0	1	0	0	1
1	0	0	1	0
0	0	1	0	1
0	1	1	1	0
1	0	1	0	0

其中，high_bit表示计数过程中的高位，low_bit是对应的低位，input表示下一个输入，high_bit_output是高位对应的输出，low_bit_output是低位对应的输出。前三行对应输入为0的情况，此时输[illegible]变化；后三行对应输入为1的情况，需要注意的就是最后一行，10加1变成00。输入和输出为11的情况不会出现，未列出。

有了这个真值表，我们就可以针对高低两位利用最小项分别写出逻辑表达式：

$$\begin{aligned} low &= low \cdot \overline{high} \cdot \overline{input} + \overline{low} \cdot \overline{high} \cdot input \\ &= \overline{high} \cdot (low \cdot \overline{input} + \overline{low} \cdot input) \\ &= \overline{high} \cdot (low \oplus input) \\ high &= high \cdot \overline{low} \cdot \overline{input} + \overline{high} \cdot low \cdot input \end{aligned}$$

最终的结果中low就表示出现一次的整数，因为0次、2次、3次对应的low值都是0。从这里也解释了为什么需要将11重置为00，否则就会出现两种情况low值为1。此外，这里还需要注意一点，两个式中的输入都是利用旧值计算新值，所以当我们在计算出low之后，不能用该low值计算high值，需要用旧的low值计算high值。新的代码如下：

```
int singleNumberII(int* A,int len)
{
        int low = 0, high = 0;
        for(int i = 0; i < len; i++){
                int temp_low = (low ^ A[i]) & ~high;
                high =(high&~low&~A[i])|(~high&low&A[i]);
                low=temp_low;
        }
        return low;
}
```

上述代码较最原始的代码优化很多，空间复杂度和时间复杂度都有明显下降，但是利用了一个局部变量，显得非常不美观。这个代码很像交换两个数时的代码，为了交换两个数，常规方法是引入一个局部变量，然后三次赋值操作。为了避免引入局部变量，一种优化是通过三次直接的位运算实现。在此我们也对上面的代码进行类似的优化，将局部变量删除。该怎么优化呢？我要放大招了！

low的计算保持不变，当我们计算完low之后，high的计算公式依赖的是旧的low值，我们设法将依赖旧low值改为依赖新low值。修改方法就是修改真值表，将low的输出重新作为输入，构造新的真值表：

high_bit	low_bit	input	high_bit_output
0	0	0	0
0	1	0	0
1	0	0	1
0	1	1	0
0	0	1	1
1	0	1	0

上述真值表唯一的修改就是把之前真值表最后一列的输出拷贝到第二列中。然后我们针对修改后的真值表求逻辑表达式：

$$\begin{aligned} high &= high \cdot \overline{low} \cdot \overline{input} + \overline{high} \cdot \overline{low} \cdot input \\ &= \overline{low} \cdot (high \cdot \overline{input} + \overline{high} \cdot input) \\ &= \overline{low} \cdot (high \oplus input) \end{aligned}$$

看到没，直接利用low的输出计算high会简化公式，同时也无需引入局部变量，最终代码如下：

```
int singleNumberII(int* A,int len)
{
        int low = 0, high = 0;
        for(int i = 0; i < len; i++){
                low = (low ^ A[i]) & ~high;
                high = (high ^ A[i]) & ~low;
        }
        return low;
}
```

上面的代码是不是非常简洁和优雅！背后其实有非常坚实的理论基础。采用真值表的方法不光优雅，还具有非常好的扩展性。假设问题改为：只有一个数出现两次，其余出现三次，我们只需要返回high即可。又假如：有一个数出现一次或者两次，其余出现三次，我们只需要返回low|high即可。此外，不只是出现三次，出现五次、七次也可以用构造真值表的方法来解决，只需要增加输入位数即可。即使是最原始的问题，我们也可以用这种方法解决，只需要构造low和input的真值表即可，你会发现构造的真值表正好就是异或的真值表！