[面试题：找出数组中只出现一次的2个数（异或的巧妙应用）（出现3次）](http://www.cnblogs.com/youxin/p/3349834.html)

2013-10-02 23:49 by youxin, 7423 阅读, 0 评论, [收藏](https://www.cnblogs.com/youxin/p/3349834.html), [编辑](https://i.cnblogs.com/EditPosts.aspx?postid=3349834)

题目：一个整型数组里除了两个数字之外，其他的数字都出现了两次。请写程序找出这两个只出现一次的数字。要求时间复杂度是O(n)，空间复杂度是O(1)。

分析：这是一道很新颖的关于位运算的面试题。

首先我们考虑这个问题的一个简单版本：一个数组里除了一个数字之外，其他的数字都出现了两次。请写程序找出这个只出现一次的数字。

这个题目的突破口在哪里？题目为什么要强调有一个数字出现一次，其他的出现两次？我们想到了异或运算的性质：任何一个数字异或它自己都等于0。也就是说，如果我们从头到尾依次异或数组中的每一个数字，那么最终的结果刚好是那个只出现依次的数字，因为那些出现两次的数字全部在异或中抵消掉了。

有了上面简单问题的解决方案之后，我们回到原始的问题。如果能够把原数组分为两个子数组。在每个子数组中，包含一个只出现一次的数字，而其他数字都出现两次。如果能够这样拆分原数组，按照前面的办法就是分别求出这两个只出现一次的数字了。

我们还是从头到尾依次异或数组中的每一个数字，那么最终得到的结果就是两个只出现一次的数字的异或结果。因为其他数字都出现了两次，在异或中全部抵消掉了。由于这两个数字肯定不一样，那么这个异或结果肯定不为0，也就是说在这个结果数字的二进制表示中至少就有一位为1。我们在结果数字中找到第一个为1的位的位置，记为第N位。现在我们以第N位是不是1为标准把原数组中的数字分成两个子数组，第一个子数组中每个数字的第N位都为1，而第二个子数组的每个数字的第N位都为0。

现在我们已经把原数组分成了两个子数组，每个子数组都包含一个只出现一次的数字，而其他数字都出现了两次。因此到此为止，所有的问题我们都已经解决。

基于上述思路，我们不难写出如下代码：

[复制代码](javascript:void(0);)

///////////////////////////////////////////////////////////////////////

// Find two numbers which only appear once in an array

// Input: data - an array contains two number appearing exactly once,

// while others appearing exactly twice

// length - the length of data

// Output: num1 - the first number appearing once in data

// num2 - the second number appearing once in data

///////////////////////////////////////////////////////////////////////

void FindNumsAppearOnce(int data[], int length, int &num1, int &num2)

{

if (length < 2)

return;

// get num1 ^ num2

int resultExclusiveOR = 0;

for (int i = 0; i < length; ++ i)

resultExclusiveOR ^= data[i];

// get index of the first bit, which is 1 in resultExclusiveOR

unsigned int indexOf1 = FindFirstBitIs1(resultExclusiveOR);

num1 = num2 = 0;

for (int j = 0; j < length; ++ j)

{

// divide the numbers in data into two groups,

// the indexOf1 bit of numbers in the first group is 1,

// while in the second group is 0

if(IsBit1(data[j], indexOf1))

num1 ^= data[j];

else

num2 ^= data[j];

}

}

///////////////////////////////////////////////////////////////////////

// Find the index of first bit which is 1 in num (assuming not 0)

///////////////////////////////////////////////////////////////////////

unsigned int FindFirstBitIs1(int num)

{

int indexBit = 0;

while (((num & 1) == 0) && (indexBit < 32))

{

num = num >> 1;

++ indexBit;

}

return indexBit;

}

///////////////////////////////////////////////////////////////////////

// Is the indexBit bit of num 1?

///////////////////////////////////////////////////////////////////////

bool IsBit1(int num, unsigned int indexBit)

{

num = num >> indexBit;

return (num & 1);

}

[复制代码](javascript:void(0);)

示例：

01 10 11 11 100 100  异或结果：11

分组：

01 11 11 异或num1=01

10 100 100 异或num2=10.

成功找到num1和num2.

 求最低位1：

int get\_first\_bit(int num)  
{  
return num&~(num - 1);  
}

求一个数最低位1的个数还有多种方法（编程之美提到过）。

参考：剑指offerhttp://zhedahht.blog.163.com/blog/static/2541117420071128950682/

相似题:

题目为：**给你1-1000个连续自然数，然后从中随机去掉两个，再打乱顺序，要求只遍历一次，求出被去掉的两个数。**

**（基本跟上面的题一样）**

解法1：使用异或。

说说异或的两个特性：顺序无关 / 对一个数异或两次等于没有异或。顺序无关就是说异或的元素可以随意交换顺序，而不会影响结果。异或两次可以理解为+x和-x。

首先，这两个数组(打乱前和打乱后)各自异或，也就是1^2^…^1000，得到两个异或值。再对这两个异或值进行一次异或，这样就得到了x^y的指(重复部分互相抵消了)。

**获取计算出的异或值的1所在的位置，并继续异或**

因为x和y是两个不同的整数，所以这两个数的异或结果，转化为二进制的话，一定在某位是1，假设在第3位。也就是说如果把原始数组按第3位是否为0进行划分，就可以分成两个数组，每个数组各包含一个被抽取的数。如果打乱后的数组也按这个规则划分为两个数组，这样就得到了4个数组，其中两组是第3位为0，另外两组是第3位为1。把第3位为0的两个数组所有元素进行异或就能得到被抽取的一个数，同理也就能获得另外一个被抽取的数，于是问题解决。

举例：4个数: 01 10 11 100

我们假设去掉01和10.异或结果为11.按第0为是否为0.

01 11                10    100

 11                     100

把左边的异或得到01

右边的得到10，问题解决。

另一种方法：用方程求解。

m = ( 1 + 2 + ...+ 1000) - (998 个的和) x + y

n = ( 1 \* 2 \* .... \* 1000) / ( 998 个的积)x \* y

经公式计算：

x =  sqart( pow( m , 2 ) / 4 - n ) + m /2

y = m - x

代码测试：

double x = 3 ;

double y = 39 ;

double m = x + y ;

double n = x \* y ;

x = Math.sqrt( m \* m / 4d - n ) + m / 2 ;

y = m - x ;

System.out.println( x );

System.out.println( y );

**另一道相似的题:**

**找数字分析**

**原题**

数组A中，除了某一个数字x之外，其他数字都出现了三次，而x出现了一次。请给出最快的方法，找到x。

**分析**

乍一看这个题目，不少同学立马给出了答案：异或。但举个例子，就会发现，异或是行不通的，一般的方法是利用异或的的如下特性：

* A xor A = 0
* A xor 0 = A

但是这个题目中，数字都是奇数个的，直接采用之前类似题目的异或方法，已经不合适了。

除此之外，我们还可能想到如下的方法：

* 采用hashmap，时间复杂度O(n)，空间复杂度O(n)
* 对数组A进行排序，然后在遍历一次，时间复杂度O(nlogn)，空间复杂度O(1) 这个方法还可以。

是否还有一些效果更好的方法呢？这一类的题目，即使简单的异或不能解决，也可以从二进制位、位操作方面去考虑，总之这样的大方向是不会错的。

题目中，如果数组中的元素都是三个三个出现的，那么从二进制表示的角度，每个位上的1加起来，应该可以整除3。如果有一个数x只出现一次，会是什么情况呢？

* 如果某个特定位上的1加起来，可以被3整除，说明对应x的那位是0，因为如果是1，不可能被3整除
* 如果某个特定位上的1加起来，不可以被3整除，说明对应x的那位是1

根据上面的描述，我们可以开辟一个大小为32的数组，第0个元素表示，A中所有元素的二进制表示的最低位的和，依次类推。最后，再转换为十进制数即可。这里要说明的是，用一个大小为32的整数数组表示，同样空间是O(1)的。

程序实现:

[复制代码](javascript:void(0);)

#include<iostream>

using namespace std;

void set(int& a,int i) { a |= (1<< (i & 0x1F));}

void clr(int& a,int i) { a &= ~(1<<(i & 0x1f));}

//除了某一个数字x之外，其他数字都出现了三次，而x出现了一次

void find(int a[],int n)

{

int m[32];

for(int i=0;i<32;i++)

m[i]=0;

for(int i=0;i<32;i++)

{

for(int j=0;j<n;j++)

{

int bit=a[j]&1;//&相当于mod 2

m[i]+=bit;

a[j] >>=1;

}

}

/\*

for(int i=0;i<32;i++)

cout<<m[i]<<ends;

cout<<endl;

\*/

int result=0;

for(int i=0;i<32;i++)

{

if(m[i]%3!=0)

set(result,i);

}

cout<<"结果为"<<result<<endl;

}

int main()

{

int a[]={1,2,2,2,3,3,3};

int n=sizeof(a)/sizeof(a[0]);

find(a,n);

/\*

int b=2;

set(b,0);

clr(b,1);

cout<<b<<endl;

\*/

}

[复制代码](javascript:void(0);)

函数

void set(int& a,int i) { a |= (1<< (i & 0x1F));} 把a第i位置为1；

void clr(int& a,int i) { a &= ~(1<<(i & 0x1f));} 把a的第i位清0.  
参考了以前的位图排序:http://www.cnblogs.com/youxin/p/3304667.html)

不过这里申请了一个数组的空间，如果这个是不被允许的呢？

参考：http://www.ituring.com.cn/article/56178

**题目：一个数组中有三个数字a、b、c只出现一次，其他数字都出现了两次。请找出三个只出现一次的数字。**

（与最前面的一题不同，前面是2个不同，现在是3个）

（要求空间为O(1)，所以用hash判断是否重复这种方法不管用了）

分析：在博客<http://zhedahht.blog.163.com/blog/static/2541117420071128950682/>中我们讨论了如何在一个数组中找出两个只出现一次的数字。在这道题中，如果我们能够找出一个只出现一次的数字，剩下两个只出现一次的数字就很容易找出来了。

如果我们把数组中所有数字都异或起来，那最终的结果（记为x）就是a、b、c三个数字的异或结果（x=a^b^c）。其他出现了两次的数字在异或运算中相互抵消了。

我们可以证明异或的结果x不可能是a、b、c三个互不相同的数字中的任何一个。我们用反证法证明。假设x等于a、b、c中的某一个。比如x等于a，也就是a=a^b^c。因此b^c等于0，即b等于c。这与a、b、c是三个互不相同的三个数相矛盾。

由于x与a、b、c都各不相同，因此x^a、x^b、x^c都不等于0。

我们定义一个函数f(n)，它的结果是保留数字n的二进制表示中的最后一位1，而把其他所有位都变成0。比如十进制6表示成二进制是0110，因此f(6)的结果为2（二进制为0010）。f(x^a)、f(x^b)、f(x^c)的结果均不等于0。

接着我们考虑f(x^a)^f(x^b)^f(x^c)的结果。由于对于非0的n，f(n)的结果的二进制表示中只有一个数位是1，因此f(x^a)^f(x^b)^f(x^c)的结果肯定不为0。这是因为对于任意三个非零的数i、j、k，f(i)^f(j)的结果要么为0，要么结果的二进制结果中有两个1。不管是那种情况，f(i)^f(j)都不可能等于f(k)，因为f(k)不等于0，并且结果的二进制中只有一位是1。

于是f(x^a)^f(x^b)^f(x^c)的结果的二进制中至少有一位是1。假设最后一位是1的位是第m位。那么x^a、x^b、x^c的结果中，有一个或者三个数字的第m位是1。

接下来我们证明x^a、x^b、x^c的三个结果第m位不可能都是1。还是用反证法证明。如果x^a、x^b、x^c的第m位都是1，那么a、b、c三个数字的第m位和x的第m位都相反，因此a、b、c三个数字的第m位相同。如果a、b、c三个数字的第m位都是0，x=a^b^c结果的第m位是0。由于x和a两个数字的第m位都是0，x^a结果的第m位应该是0。同理可以证明x^b、x^c第m位都是0。这与我们的假设矛盾。如果a、b、c三个数字的第m位都是1，x=a^b^c结果的第m位是1。由于x和a两个数字的第m位都是1，x^a结果的第m位应该是0。同理可以证明x^b、x^c第m位都是0。这还是与我们的假设矛盾。

因此x^a、x^b、x^c三个数字中，只有一个数字的第m位是1。于是我们找到了能够区分a、b、c三个数字的标准。这三个数字中，只有一个数字满足这个标准，而另外两个数字不满足。一旦这个满足标准数字找出来之后，另外两个数字也就可以找出来了。

[复制代码](javascript:void(0);)

void getThreeUnique(vector<int>& numbers, vector<int>& unique)

{

if(numbers.size() < 3)

return;

int xorResult = 0;

vector<int>::iterator iter = numbers.begin();

for(; iter != numbers.end(); ++iter)

xorResult ^= \*iter;

int flags = 0;

for(iter = numbers.begin(); iter != numbers.end(); ++iter)

flags ^= lastBitOf1(xorResult ^ \*iter);

flags = lastBitOf1(flags);

// get the first unique number

int first = 0;

for(iter = numbers.begin(); iter != numbers.end(); ++iter)

{

if(lastBitOf1(\*iter ^ xorResult) == flags)

first ^= \*iter;

}

unique.push\_back(first);

// move the first unique number to the end of array

for(iter = numbers.begin(); iter != numbers.end(); ++iter)

{

if(\*iter == first)

{

swap(\*iter, \*(numbers.end() - 1));

break;

}

}

// get the second and third unique numbers

getTwoUnique(numbers.begin(), numbers.end() - 1, unique);

}

int lastBitOf1(int number)

{

return number & ~(number - 1);

}

void getTwoUnique(vector<int>::iterator begin, vector<int>::iterator end, vector<int>& unique)

{

int xorResult = 0;

for(vector<int>::iterator iter = begin; iter != end; ++iter)

xorResult ^= \*iter;

int diff = lastBitOf1(xorResult);

int first = 0;

int second = 0;

for(vector<int>::iterator iter = begin; iter != end; ++iter)

{

if(diff & \*iter)

first ^= \*iter;

else

second ^= \*iter;

}

unique.push\_back(first);

unique.push\_back(second);

}

[复制代码](javascript:void(0);)

上文中getThreeUnique从数组中找出三个只出现一次的数字，而getTwoUnique从数组中找出两个只出现一次的数字。lastBitOf1实现分析中的函数f(n)的功能，它只保留数字n的二进制表示中的最后一位1，而把其他所有位都变成0。

在函数getThreeUnique中，我们通过第一个for循环把a、b、c三个数字异或的结果保存到xorResult中，接着在第二个for循环中求出f(x^a)^f(x^b)^f(x^c)并保存到变量flags中。在语句flags=lastBitOf1(flags)求出f(x^a)^f(x^b)^f(x^c)结果的二进制中最后一位是1的位。并根据这一数位求出第一个只出现一次的数字first。接着把first交换到数组的最后，并在数组的前n-1个数字中求出另外两个只出现一次的数字。

上面的简单代码版：http://blog.csdn.net/zzran/article/details/8108787

考虑给定数组中有三个单独出现一次的数字，这个会比有两个的稍微复杂。分步分析，设定这三个数为a,b,c:

（1）将数组中的数字全部异或，得到的结果x=a^b^c，但是x不是a，b，c中的其中一个，假设x=a，那么b^c=0说明b=c，与题目给定的条件矛盾。

(2)设定f(n)可以像2中的那样，从低位开始，找到第一个bit为1的位置，f(x^a),f(x^b),f(x^c)得到的值肯定都不为0，因为x^a,x^b,x^c本身就不为0。f(x^a)^f(x^b)^f(x^c)结果不为0。因为f(x^a)^f(x^b)的结果中可能为0，也可能有两个bit为1。如果假设f(x^c)的结果bit为1的位置与f(x^a)^f(x^b)的其中一个重合，则f(x^a)^f(x^b)^f(x^c)结果中只有1个bit为1，如果不重合的话那么有3个bit位为1。

(3)这便可以推断出f(x^a)^f(x^b)^f(x^c)中至少有一个bit位为1。假设从低位到高位的第mbit位为1.那么可以得出结论x^a,x^b,x^c中有一个或者三个的第m位为1（不可能有两个，因为有两个的话，异或的结果就为0了）。

（4）证明，x^a,x^b,x^c中只有一个第m-bit位为1.假设他们的第m位都为1，那么x的第m位为0，但是x=a^b^c其第m位肯定为1，所以假设不成立。那么相反，假设x的第m位为1，a,b,c的第m位都为0，也不成立，因为x=a^b^c。所以综上所述x^a,x^b,x^c中只有一个第m位为1。那么这个问题就好办了。根据这个第m位找到第一个只出现一次的数字。然后剩下两个就是问题2所描述的问题。下面给出代码：

[复制代码](javascript:void(0);)

#include<stdio.h>

int get\_first\_bit(int num)

{

return num&~(num-1);

}

void get\_two\_unique\_num(int \*a,int n,int \*num1,int \*num2)

{

int result\_code=0;

for(int i=0;i<n;i++)

result\_code^=a[i];

int diff=get\_first\_bit(result\_code);

\*num1=0;

\*num2=0;

for(i=0;i<n;i++)

{

if(a[i]&diff)

{

(\*num1)^=a[i];

}

else

{

(\*num2)^=a[i];

}

}

}

void get\_three\_unique\_num(int \*a,int n,int \*num1,int \*num2,int \*num3)

{

int result\_code=0;

for(int i=0;i<n;i++)

result\_code^=a[i];

int flag=0;

for(i=0;i<n;i++)

flag^=get\_first\_bit(result\_code^a[i]);

flag=get\_first\_bit(flag);

\*num1=0;

for(i=0;i<n;i++)

{

if(get\_first\_bit(result\_code^a[i])==flag)

{

(\*num1)^=a[i];

}

}

for(i=0;i<n;i++)

{

if(a[i]==(\*num1))

{

int temp=a[i];

a[i]=a[n-1];

a[n-1]=temp;

break;

}

}

get\_two\_unique\_num(a,n-1,num2,num3);

}

void main()

{

int a[]={2,2,4,4,6,6,3,5,7};

int num1,num2,num3;

get\_three\_unique\_num(a,sizeof(a)/sizeof(int),&num1,&num2,&num3);

printf("%d\t%d\t%d\n",num1,num2,num3);

}