数学专栏课外加餐（二）讲位操作的三个应用实例

你好，我是黄申。欢迎来到第二次课外加餐时间。

位操作的应用实例

留言里很多同学对位操作比较感兴趣，我这里通过计算机中的位操作的几个应用，来帮你理解位操作。

1.验证奇偶数

在第2节里，我提到了，奇偶数其实也是余数的应用。编程中，我们也可以用位运算来判断奇偶数。

仔细观察，你会发现偶数的二进制最后一位总是0，而奇数的二进制最后一位总是1，因此对于给定的某个数字，我们可以把它的二进制和数字1的二进制进行按位“与”的操作，取得这个数字的二进制最后一位，然后再进行判断。

我这里写了一段代码，比较了使用位运算和模运算的效率，我统计了进行1亿次奇偶数判断，使用这两种方法各花了多少毫秒。如果在你的机器上两者花费的时间差不多，你可以尝试增加统计的次数。在我的机器上测试下来，同样次数的奇偶判断，使用位运算的方法耗时明显更低。

```
public class Lesson1_append1 {

 public static void main(String[] args) {

  int even_cnt = 0, odd_cnt = 0;
  long start = 0, end = 0;

  start = System.currentTimeMillis();
  for (int i = 0; i < 100000000; i++) {

   if((i & 1) == 0){
       even_cnt ++;
   }else{
       odd_cnt ++;
   }

  }
  end = System.currentTimeMillis();
  System.out.println(end - start);
  System.out.println(even_cnt + " " + odd_cnt);

  even_cnt = 0;
  odd_cnt = 0;
  start = 0;
  end = 0;

  start = System.currentTimeMillis();
  for (int i = 0; i < 100000000; i++) {

   if((i % 2) == 0){
       even_cnt ++;
   }else{
       odd_cnt ++;
   }

  }
  end = System.currentTimeMillis();
  System.out.println(end - start);
  System.out.println(even_cnt + " " + odd_cnt);

 }
}
```

2.交换两个数字

你应该知道，要想在计算机中交换两个变量的值，通常都需要一个中间变量，来临时存放被交换的值。不过，利用异或的特性，我们就可以避免这个中间变量。具体的代码如下：

```
x = (x ^ y);
y = x ^ y;
x = x ^ y;
```

把第一步代入第二步中, 可以得到：

```
y = (x ^ y) ^ y = x ^ (y ^ y) = x ^ 0 = x
```

把第一步和第二步的结果代入第三步中，可以得到：

```
x = (x ^ y) ^ x = (x ^ x) ^ y = 0 ^ y = y
```

这里用到异或的两个特性，第一个是两个相等的数的异或为0，比如x^x= 0；第二个是任何一个数和0异或之后，还是这个数不变，比如0^y=y。

3.集合操作

集合和逻辑的概念是紧密相连的，因此集合的操作也可以通过位的逻辑操作来实现。

假设我们有两个集合{1, 3, 8}和{4, 8}。我们先把这两个集合转为两个8位的二进制数，从右往左以1到8依次来编号。

如果某个数字在集合中，相应的位置1，否则置0。那么第一个集合就可以转换为10000101，第二个集合可以转换为10001000。那么这两个二进制数的按位与就是10000000，只有第8位是1，代表了两个集合的交为{8}。而这两个二进制数的按位或就是10001101，第8位、第4位、第3位和第1位是1，代表了两个集合的并为{1, 3, 4, 8}。

说到这里，不禁让我想起Elasticsearch的BitSet。我曾经使用Elasticsearch这个开源的搜索引擎来实现电商平台的搜索。

当时为了提升查询的效率，我使用了Elasticsearch的Filter查询。我研究了一下这个Filter查询的原理，发现它并没有考虑各种文档的相关性得分，因此它可以把文档匹配关键字的情况，转换成了一个BitSet。

你可以把BitSet想成一个巨大的位数组。每一位对应了某篇文档是否和给定的关键词匹配，如果匹配，这一位就置1，否则就置0。每个关键词都可以拥有一个BitSet，用于表示哪些文档和这个关键词匹配。那么要查看同时命中多个关键词的文档有哪些，就是对多个BitSet求交集。利用上面介绍的按位与，这点是很容易实现的，而且效率相当之高。

二分查找时的两个细节

第3节我介绍了迭代法，并讲解了相关的代码实现。其中，有两个细节我在这里补充说明一下。

第一个是关于**中间值的计算**。我优化了两处代码，分别是Lesson3_2的第16行和Lesson3_3的第22行。

其中，Lesson3_2的第16行由原来的：

```
double middle = (min + max) / 2;
```

改为：

```
double middle = min + (max - min) / 2;
```

Lesson3_3的第22行由原来的：

```
int middle = (left + right) / 2;
```

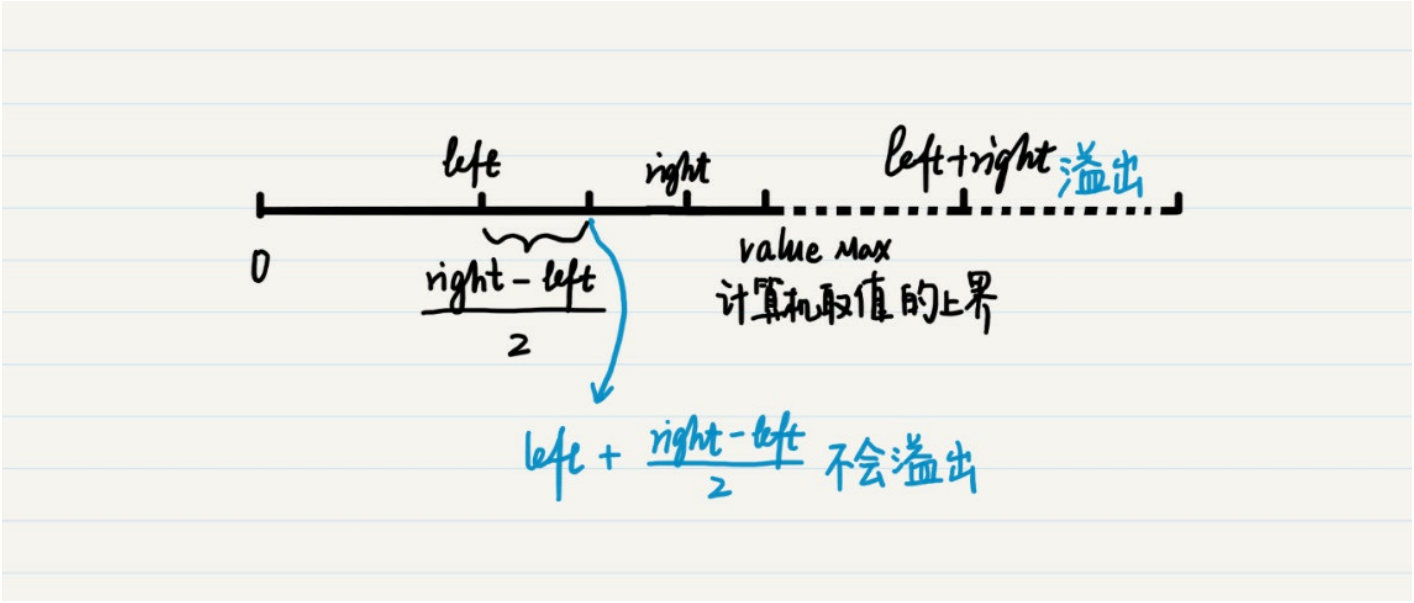
改为：

```
int middle = left + (right - left) / 2;
```

这两处改动的初衷都是一样的，是为了避免溢出。在第一篇加餐中，介绍负数的加法时，我已经解释了什么是溢出。那这里为什么会发生溢出呢？我以第二处代码为例来讲解下。

从理论上来说，(left+right)/2=left+(right-left)/2。可是，我们之前说过，计算机系统有自身的局限性，无论是何种数据类型，都有一个上限或者下限。一旦某个数字超过了这些限定，就会发生溢出。

对于变量left和right而言，在定义的时候都指定了数据类型，因此不会超出范围。可是，left+right的和就不一定了。从下图可以看出，当left和right都已经很接近某个数据类型的最大值时，两者的和就会超过这个最大值，发生上溢出。这也是为什么最好不用通过(left+right)/2来求两者的中间值。



那么为什么left + (right -left)/2就不会溢出呢？首先，right是没有超过最大值的，那么(right -left)/2自然也就没有超过范围，即使left加上了(right -left)/2，也不会超过right的值，所以运算的整个过程都不会产生溢出。

第二个是关于误差百分比和绝对误差。在Lesson3_2中有这么一行：

```
double delta = Math.abs((square / n) - 1);
```

这里我使用了误差的百分比，也就是误差值占输入值n的比例。其实绝对误差也是可以的，不过我在这里考虑了n的大小。比如，如果n是一个很小的正整数，比如个位数，那么误差可能要精确到0.00001。但是如果n是一个很大的数呢？比如几个亿，那么精确到0.00001可能没有多大必要，也许精确到0.1也就可以了。所以，使用误差的百分比可以避免由于不同的n，导致的迭代次数有过大差异。

由于这里n是大于1的正整数，所以可以直接拿平方值square去除以n。否则，我们要单独判断n为0的情况，并使用绝对误差。

关于迭代法、数学归纳法和递归

从第3节到第6节，我连续介绍了迭代法、数学归纳法、递归。这些概念之间存在相互联系，又不完全一样，很多同学对此也有一些疑惑。所以，这里我来帮你梳理一下。

迭代法和递归都是通过不断反复的步骤，计算数值或进行操作的方法。迭代一般适合正向思维，而递归一般适合逆向思维。而递归回溯的时候，也体现了正向递推的思维。它们本身都是抽象的流程，可以有不同的编程实现。

对于某些重复性的计算，数学归纳法可以从理论上证明某个结论是否成立。如果成立，它可以大大节约迭代法中数值计算部分的时间。不过，在使用数学归纳法之前，我们需要通过一些数学知识，假设命题，并证明该命题成立。

对于那些无法使用数学归纳法来证明的迭代问题，我们可以通过编程实现。这里需要注意的是，广义上来说，递归也是迭代法的一种。不过，在计算机编程中，我们所提到的迭代是一种具体的编程实现，是指使用循环来实现的正向递推，而递归是指使用函数的嵌套调用来实现的逆向递推。当然，两种实现通常是可以相互转换的。

循环的实现很容易理解，对硬件资源的开销比较小。不过，循环更适合“单线剧情”，例如计算2^n，n!，1+2+3+…+n等等。而对于存在很多“分支剧情”的复杂案例而言，使用递归调用更加合适。

利用函数的嵌套调用，递归编程可以存储很多中间变量。我们可以很轻松地跟踪不同的分支，而所有这些对程序员基本是透明的。如果这时使用循环，我们不得不自己创建并保存很多中间变量。当然，正是由于这个特性，递归比较消耗硬件资源。

递归编程本身就体现了分治的思想，这个思想还可以延伸到集群的分布式架构中。最近几年比较主流的MapReduce框架也体现了这种思想。

综合上面说的几点，你可以大致遵循这样的原则：

- 如果一个问题可以被迭代法解决，而且是有关数值计算的，那你就看看是否可以假设命题，并优先考虑使用数学归纳法来证明；
- 如果需要借助计算机，那么优先考虑是否可以使用循环来实现。如果问题本身过于复杂，再考虑函数的嵌套调用，是否可以通过递归将问题逐级简化；
- 如果数据量过大，可以考虑采用分治思想的分布式系统来处理。

最后，给你留一道思考题吧。

在1到n的数字中，有且只有唯一的一个数字m重复出现了，其它的数字都只出现一次。请把这个数字找出来。提示：可以充分利用异或的两个特性。

好了，前面6讲的补充内容就到这里了。欢迎你留言给我。你也可以点击“请朋友读”，把今天的内容分享给你的好友，和他一起精进。

精选留言

Jerry银银

我的天，昨天才为老师的加餐点过赞，今天又来一篇干货。谢谢老师，看了这两篇加餐，心里的很多疑惑被解除了。买老师的专栏，值了。

——

思考题：

需要考虑不同的数量级，分两种情况：

1. 内存能容纳这n个数

方法1：暴力查找，两层循环遍历，时间复杂度为O(n^2)，空间复杂度为O(1)

方法2：用快排先进行排序，然后遍历一次，比较前一个数和后一个数，若相等，则查找完成，时间复杂度O(nlogn)，空间复杂度为O(1)

方法3：利用hash表(或set)，进行一次遍历，同时将遍历到的数放入hash表，放入之前判断hash表是否存在，若存在，则找到了重复的数，时间复杂度为O(n)，空间复杂度为O(n)

方法4：使用位向量，遍历给到的n个数，对于出现的数，将对应位标记为1，如果已经是1则查找成功，时间复杂度为O(n)，空间复杂度为(n)，这种方法类似方法3，虽然渐进的空间复杂度和方法3相同，但是其实小很多很多，毕竟只要用1bit就能表示有或无

2. 内存无法容纳给到的n个数

依然可以用上述方法4来解决，其它的方法有的不能用，有的效率不高。

2018-12-26 07:57

作者回复

感谢支持，思考题分析的很透彻，各种情况都考虑到了

2018-12-26 11:58

科哥

根据异或的两个特点，任何两个相同的数异或的结果都为0，任何数与0异或都为这个数，因此将所有的数依次异或得到的结果就是除了两个重复数的所有数的异或结果，假设为T。而将1到n依次异或的结果为T^重复数。因此，重复数=T^T^重复数。即:所有数异或的结果再异或1到n所有数异或的结果

2018-12-26 10:06

作者回复

很好的思路

2018-12-26 11:41

李嘉鹏

看了大家留言，1－n必须是连续以1递增才有简化解的吧。从原题并未审出这一点。

2018-12-28 07:50

mickey

/**

在1到n的数字中，有且只有唯一的一个数字m重复出现了，其它的数字都只出现一次。

请把这个数字找出来。

提示：可以充分利用异或的两个特性。

*/

```
public class LessonE02_2 {
public static void main(String[] args) {
int[] arr = { 6, 3, 9, 5, 4, 8, 2, 5, 7, 1 };

int temp = arr[0];
int max = 0;
for (int i = 1; i < arr.length; i++) {
temp ^= arr[i];
max = max < arr[i] ? arr[i] : max;
}
int t = 1;
for (int i = 2; i <= max; i++) {
t ^= i;
}
System.out.println(t ^ temp);
}
}
```

题目不够严谨，m 重复偶数次才能用位运算吧。

2018-12-26 13:39

指间砂的宿命

将所有结果异或再和1到n的不重复结果异或，最后剩余的值就是重复值，真的好神奇，这种异或用法

2018-12-26 09:35

作者回复

嗯 异或的妙用

2018-12-26 11:48

樊少皇

思考题：

```
public static int getSpecialNum(int[] oArr, int n){
int result = 0;
for(int i = 0; i < oArr.length; i++){
```

result = (result^oArr[i]);
}
for(int j = 1; j <= n; j++){
result = result^j;
}
return result;
}

最开始不太理解。后来明白题意应该是说总共有n+1或者更多个数字，这些数字都在1--n内，并且除数字m外，其余的数字有且只有一个。举例：{1,2,3,4,5,5,6,7}满足条件；{1,2,3,5,5,5,6,7}不满足条件。

2018-12-26 18:01

胡鹏

看到 Brian Wang 的回答, 您说了正解, 我才想明白:
推到应该是:
原始数据: 1,2...m,m,...n (是否有序对此题不重要)
所有数字: 1,2,...m,...n
因为 x^x = 0
令a = 1^2...^m...^n
b = 1^2...^m^m...^n
则有: a^b = (1^2...^m...^n)^(1^2...^m...^n)^m = 0^m = m

2018-12-30 18:00

作者回复

是的

2018-12-31 05:01

夏飞

假设我们有两个集合{1, 3, 8}和{4, 8}。我们先把这两个集合转为两个 8 位的二进制数，从右往左以 1 到 8 依次来编号。

如果某个数字在集合中，相应的位置 1，否则置 0。那么第一个集合就可以转换为 10000101，第二个集合可以转换为 10001000。

怎么转的？没看懂

2018-12-26 09:45

作者回复

10000101最高位（第8位）的1表示集合中的8，第3位的1表示集合中的3，最低位的1表示集合中的1，以此类推。然后两个集合的交集就专为两个二进制数的按位与

2018-12-26 11:48

Brian Wang

思考题：对于有的全部数字进行异或再和 1-n 这 n 个数字进行异或，最终得出的结果就是 m

2018-12-26 08:30

作者回复

正解

2018-12-26 11:50

蜉蝣

如果给出的数字不连续，Python中不妨这样使用：

```python
from itertools import chain

nums_list = [1, 2, 10, 8, 2, 3]
nums_set = set(nums_list)
```

start = 0 # 任何数与 0 异或得到自己，所以作为初始值使用
for num in chain(nums_list, nums_set):
start = start ^ num

print(start)
```

2019-01-08 13:49

作者回复

正确

2019-01-14 02:11

bnhjk76

集合｛1，3，8｝最大的数是8所以用8位数？那如果最大的数是999999999...那这个2进制的位数会很大，这个时候也用这个方法进行运算吗？

2019-01-04 08:53

作者回复

可以把大的数切分为不同的组，例如第一个32位二进制表示前32个元素，第二个二进制表示第33到第64的元素

2019-01-05 08:42

梦开始的地方

对于有的全部数字进行异或再和 1-n 这 n 个数字进行异或，最终得出的结果就是 m
老师，我对于两全部数字异或，在和1-n异或，就能得出m不太懂

2018-12-27 23:57

anil

请教老师，用异或交换两个变量值感觉不太懂：
第一步代入第二步时，y已经=x了，
再把第二步代入第三步，此时y的值已经是x，怎么还能利用它把原y值传给x呢？
感觉还是要临时变量做过渡啊？

2018-12-26 11:22

作者回复

因为此时新的x值还是x^y，而新y已经是原来的x，两种异或，就是y了

2018-12-26 11:40

风轨

不等关系是“最没用”的关系，没有传递性，更没有序性。
如果没有额外空间，直接暴力比较，时间复杂度O(n平方); 如果这n个数字本身是有序的，需要时间复杂度O(n)，排序时间复杂度O(nlog(n))。
如果额外空间充足，在数据聚集度较高甚至连续时，可以使用桶，时间复杂度O(n);如果数据很分散，数据范围远远大于数据量，可以考虑用桶加hash，时间复杂度O(n)，但需要考虑hash碰撞问题。

2018-12-26 09:28

作者回复

很好的总结！

2018-12-26 11:49

拉欧

集合中的数字和1到n的数字组成的2n+1个数，只有m出现了3次，其他的数是2次，全部亦或一遍之后就剩下了m

2018-12-26 09:16

作者回复

正解

2018-12-26 11:49

Brian Wang

思考题：把所有的数字进行异或，结果就是要的答案

2018-12-26 08:28

谭大码畜。

思考题：

public static void findRepeatedNumber2(int[] nums) {
```

```
// TODO 排序，将 nums 中的最大数放入最右边。
// 以 nums 中的最大值 - 1 作为数组长度。
int[] map = new int[nums[nums.length - 1]];
// nums 中所有数(包含重复数)异或的结果。
int res = 0;
for (int i = 0; i < nums.length; i ++) {
int index = nums[i] & (nums[nums.length - 1] - 1);
// 这里其实可以判断是否为重复数，然后直接返回了。
map[index] = nums[i];
// 任何数与 0 进行异或，这个数都不变。
res = res ^ nums[i];
}

// nums 中出现过的数字的异或结果。
int res2 = 0;
for (int i = 0; i < map.length; i ++) {
res2 = res2 ^ map[i];
}

// (n ^ n) ^ m = 0 ^ m = m
System.out.printf("result = %d%n", res ^ res2);

}
```

写完之后才想起来，如果传入的数组中不存在重复的数，那这段代码就歇菜了。/捂脸
不过大致用异或的思路我觉得大概是这样的 :
1. 找出数组出现过的数字，并对这些数字进行异或，得到结果 A。
2. 将数组中的所有元素进行异或，得到结果 B。// 也可以看做是 A ^ m
3. A ^ B = A ^ A ^ m = 0 ^ m = m

2019-02-20 19:13

作者回复

对的

2019-02-21 01:16

Sam.张朝

1到n，数字不连续，T^(1到n异或的结果) 这样应该不可以吧?

2019-02-13 15:29

作者回复

确实需要连续，如果不连续，时间复杂度更高，异或就不是很必要了

2019-02-14 02:47

呆

这个直接所有数字求和 然后减去(1+n)n/2不就得了

2019-02-11 22:48

作者回复

如果是连续数据是可以的，不连续数据就需要其他方法。

2019-02-12 01:26

qinggeouye

思考题：
在 1 到 n 的数字中，有且只有唯一的一个数字 m 重复出现了，其它的数字都只出现了一次。请把这个数字找出来。提示：可

以充分利用异或的两个特性。

题意：1 到 n 的数字，从 1 开始，依次递增直到 n，其中 m 重复出现。
arr_a = [1, 2, 3, ..., m, m, m, ..., n-1, n] # 原数组, 与顺序无关
arr_b = [1, 2, 3, ..., m, ..., n-1, n] # 1 到 n 数字不重复的数组，与顺序无关

依次遍历 arr_a 中每个数字，异或，结果记为 res1
依次遍历 arr_b 中每个数字，异或，结果记为 res2
则 m = res1 ^ res2

2019-02-07 21:02

作者回复

你说的方法是可以的，不过更简单的是直接将这n个数字异或，而且1-n也不必连续。

2019-02-08 00:42