位运算技巧

操作符：

&（与）--都为1则1，否则0

～（非）--取反

｜（或）--有1则为1，全零为0

^(异或) --相同为0，不同为1

<< (左移)：向左进行移位操作，高位丢弃，低位补 0

```
int a = 8;
a << 3;
移位前：0000 0000 0000 0000 0000 0000 0000 1000
移位后：0000 0000 0000 0000 0000 0000 0100 0000
```

>> 右移运算: 向右进行移位操作，对无符号数，高位补 0，对于有符号数，高位补符号位

```
unsigned int a = 8;
a >> 3;
移位前：0000 0000 0000 0000 0000 0000 0000 1000
移位后：0000 0000 0000 0000 0000 0000 0000 0001

int a = -8;
a >> 3;
移位前：1111 1111 1111 1111 1111 1111 1111 1000
移位前：1111 1111 1111 1111 1111 1111 1111 1111
```

常见位运算问题

- 数 a 向右移一位，相当于将 a 除以 2；数 a 向左移一位，相当于将 a 乘以 2

```
int a = 2;
a >> 1; ---> 1
a << 1; ---> 4
```

- 位操作交换两数

```
//普通操作
void swap(int &a, int &b) {
  a = a + b;
  b = a - b;
  a = a - b;
}

//位与操作
void swap(int &a, int &b) {
  a ^= b;
  b ^= a;
  a ^= b;
}
```

- 只要根据数的最后一位是 0 还是 1 来决定即可，为 0 就是偶数，为 1 就是奇数

```
if(0 == (a & 1)) {
 //偶数
}
```

- 交换符号将正数变成负数，负数变成正数

```
int reversal(int a) {
  return ~a + 1;
}
```

- 整数的绝对值是其本身，负数的绝对值正好可以对其进行取反加一求得，即我们首先判断其符号位（整数右移 31 位得到 0，负数右移 31 位得到 -1,即 0xffffffff），然后根据符号进行相应的操作

```
int abs(int a) {
  int i = a >> 31;
  return i == 0 ? a : (~a + 1);
}
// 优化
int abs2(int a) {
  int i = a >> 31;
  return ((a^i) - i);
}
```

- 位操作进行高低位交换

```
//给定一个 16 位的无符号整数，将其高 8 位与低 8 位进行交换，求出交换后的值
/*34520的二进制表示：
10000110 11011000

将其高8位与低8位进行交换，得到一个新的二进制数：
11011000 10000110
其十进制为55430
*/
unsigned short a = 34520;
a = (a >> 8) | (a << 8);
```

- 位操作进行二进制逆序

```
/*数34520的二进制表示：
10000110 11011000

逆序后则为：
00011011 01100001
它的十进制为7009
*/
unsigned short a = 34520;

a = ((a & 0xAAAA) >> 1) | ((a & 0x5555) << 1);
a = ((a & 0xCCCC) >> 2) | ((a & 0x3333) << 2);
a = ((a & 0xF0F0) >> 4) | ((a & 0x0F0F) << 4);
a = ((a & 0xFF00) >> 8) | ((a & 0x00FF) << 8);
```

- 位操作统计二进制中 1 的个数

```
count = 0
while(a){
  a = a & (a - 1);
  count++;
}
```

- 消去最后一位1

```
x & (x - 1) 用于消去x最后一位的1
```

- O(1) 时间检测整数 n 是否是 2 的幂次

```
/*
思路:
N如果是2的幂次，则满足的条件
1.N>1
2.N的二进制表示中只能有一个1，只要统计1的个数就可以了
/*
```

- 如果要将整数A转换为B，需要改变多少个bit位

```
/*
C=A^B,统计C中1个个数
*/
```

- 给定一个含不同整数的集合，返回其所有的子集

思路就是使用一个正整数二进制表示的第i位是1还是0，代表集合的第i个数取或者不取。
所以从0到2^n-1总共2^n个整数，正好对应集合的2^n个子集。

```
S = {1,2,3}
N bit Combination
0 000 {}
1 001 {1}
2 010 {2}
3 011 {1,2}
4 100 {3}
5 101 {1,3}
6 110 {2,3}
7 111 {1,2,3}
```

- 奇数次出现和偶数次出现

```
1   a ^ b ^ b = a
```

- 数组中，只有一个数出现一次，剩下都出现三次，找出出现一次的

解题思路：

因为数是出现三次的，也就是说，对于每一个二进制位，如果只出现一次的数在该二进制位为1，那么这个二进制位在全部数字中出现次数无法被3整除。
膜3运算只有三种状态：00,01,10，因此我们可以使用两个位来表示当前位%3，对于每一位，我们让Two，One表示当前位的状态，B表示输入数字的对应位，Two+和One+表示输出状态。

```
0 0 0 0 0
 0 0 1 0 1
 0 1 0 0 1
 0 1 1 1 0
 1 0 0 1 0
 1 0 1 0 0
 One+ = (One ^ B) & (~Two)
 Two+ = (~One+) & (Two ^ B)
```

- 数组中，只有两个数出现一次，剩下都出现两次，找出出现一次的

思路解析：

有了第一题的基本的思路，我们可以将数组分成两个部分，每个部分里只有一个元素出现一次，其余元素都出现两次。那么使用这种方法就可以找出这两个元素了。
不妨假设出现一个的两个元素是x，y，那么最终所有的元素异或的结果就是res = x^y。并且res!=0，那么我们可以找出res二进制表示中的某一位是1。对于原来的数组，我们可以根据这个位置是不是1就可以将数组分成两个部分。x，y在不同的两个子数组中。而且对于其他成对出现的元素，要么在x所在的那个数组，要么在y所在的那个数组。

- 获得高位都是0，地位都是1的二进制数

```
// 获取：  0000 0000 0000 0000 0000 0000  1111 1111
int N=8;
int limit= N==32? -1: (1<<n)-1
```

- 获取高位为1，低位为0的二进制数

```
// 获取：  1111 1111 0000 0000 0000 0000 0000 0000
int N=8;
int res=0;
for(int i=0;i<N;i++)
{
  res+=1<<(32-i);
}

```

- 提取二进制数最右边的1

```
// pose 1001
int index=pos &( ~pos +1);
pose=index-pos;

```