**各种位运算的使用**

=== 1. & 按位与 ===  
 & 运算通常用于二进制取位操作，例如一个数 & 1的结果就是取二进制的最末位。这可以用来判断一个整数的奇偶，二进制的最末位为0表示该数为偶数，最末位为1表示该数为奇数.

 === 2. | 按位或 ===  
 | 运算通常用于二进制特定位上的无条件赋值，例如一个数 | 1的结果就是把二进制最末位强行变成1。如果需要把二进制最末位变成0，对这个数 | 1之后再减一就可以了，其实际意义就是把这个数强行变成最接近的偶数。

=== 3. ^ 异或 ===

^ 运算通常用于对二进制的特定一位进行取反操作，因为异或可以这样定义：0和1异或0都不变，异或1则取反。

^ 运算的逆运算是它本身，也就是说两次异或同一个数最后结果不变，即(a ^ b) ^ b = a。^ 运算可以用于简单的加密，比如我想对一 MM说1314520，但怕别人知道，于是双方约定拿我的生日123456789作为密钥。1314520 ^ 123456789 = 122667981，我就把122667981告诉MM。MM再次计算122667981 ^ 123456789的值，得到1314520，于是她就明白了我的意思。

交换两个数的数值的时候就可以不用多声明一个临时变量，只需要进行x = x ^ y; y = x ^ y; x = x ^ y;对于给定的n个数值，然后给出n – 1个数值，判断哪一个数值没有出现也可以用 ^ 运算符。

=== 4. ~ 运算 ===  
 ~ 运算的定义是把内存中的0和1全部取反。使用 ~ 运算时要格外小心，你需要注意整数类型有没有符号。如果 ~ 的对象是无符号整数，那么得到的值就是它与该类型上界的差，因为取反后的数和原来的数相加后每一位都是1，然后这个数是该类型最大的数。如果 ~ 的对象是有符号整数的话，那么得到的值应该是这个数的相反数减一，因为计算一个数的相反数，就是对一个数进行取反加一。

=== 5. << 运算 ===  
 a << b就表示把a转为二进制后左移b位（在后面添b个0）。例如100的二进制为1100100，而110010000转成十进制是400，那么100 << 2 = 400。可以看出，a << b的值实际上就是a乘以2的b次方，因为在二进制数后添一个0就相当于该数的二进制转化成十进制时乘的数值再乘2，就相当于整个数值乘二，例如一百的最右边的1，本来应该乘16，右移一位后，就是乘上32。

a << 1比a \* 2更快，因为前者是直接操作内存。因此程序中乘以2的操作请尽量用左移一位来代替。

定义一些常量可能会用到 << 运算。你可以方便地用(1 << 16) - 1来表示65535， 这边要注意<< 的优先级。很多算法和数据结构要求数据规模必须是2的幂，此时可以用 << 来定义来这些数。

=== 6. << 运算 ===  
 和 << 相似，a >> b表示二进制右移b位（去掉末b位），相当于a除以2的b次方（取整），原因类似于<<，这边要注意，右移在左边补的数值和符号位相同，如果是整数，右移补0，如果是负数，右移补1。我们也经常用 >> 1来代替 / 2，比如二分查找、堆的插入操作等等。想办法用 >> 代替除法运算可以使程序效率大大提高。

Ps:

位操作的优先级一般都是比较低的，在位操作符中：

① ~ 的优先级最高，比+ - \* / %还要高，其它的位操作符优先级都比它们低，遇到要记得加好括号，~是唯一一个从右往左计算的，如果遇到多个，则从右开始计算，其它位操作符都是从左往右计算，~也是唯一一个单目运算符；

② << 和 >>；

③ &；

④ ^；

⑤ |；

**常见的二进制变换操作**

|  |  |  |
| --- | --- | --- |
| **功能** | **位操作** | **示例** |
| x / 2n | x >> n | 1111 / 21 -> 111 |
| x \* 2n | x << n | 1111 \* 21 -> 11111 |
| 2 \* x + 1 | x << 1 | 1 | 100 \* 2 + 1 -> 1001 |
| 把x的右起第n + 1位变成1 | x | 1 << n | 1000,3 -> 1100 |
| 把x最后n位变成1 | x | ((1 << n) – 1) | 1000, 3 -> 1111 |
| 对x的右起第n+ 1位取反 | x ^ 1 << n | 1111, 3 -> 1011 |
| 对x的右起n位取反 | x ^ ((1 << n) – 1) | 1101, 3 -> 1010 |
| 把x的右起第n + 1位变成0 | x & ~(1 << n) | 1111, 3 -> 0111 |
| 把x最后n位变成0 | x &~ ((1 << n) – 1) | 1, 3 -> 1000 |
| 取x的右起第n + 1位 | x >> n & 1 | 1101, 3 -> 1 |
| 取x的右起n位 | x & ((1 << n) – 1) | 1101, 3 -> 101 |
| 把x末位开始连续的1变成0 | x & (x + 1) | 1111 -> 0 |
| 把x末位开始连续的0变成1 | x | (x - 1) | 1000 -> 1111 |
| 把x的右起第一个0变成1 | x | (x + 1) | 1101 -> 1111 |
| 取x的右起第一个1 | x & -x | 1100 -> 100 |
| 把x的右起第一个1变成0 | x & ~(x & -x) | 1000 -> 0 |
| 取x末位开始连续的1 | (x ^ (x + 1)) >> 1 | 1011 -> 11 |
| 取int的绝对值 | (x ^ (x >> 31)) - (x >> 31) | 0xffffffff -> 0x1 |
| int类型x高16位和低16位交换 | (x >> 16) | (x << 8) | 0x7fffffff -> 0xffff7ffff |
| int类型x二进制逆序 | x = ((x & 0xAAAAAAAA) >> 1) | ((x & 0x55555555) << 1);  x = ((x & 0xCCCCCCCC) >> 2) | ((x & 0x33333333) << 2);  x = ((x & 0xF0F0F0F0) >> 4) | ((x & 0x0F0F0F0F) << 4);  x = ((x & 0xFF00FF00) >> 8) | ((x & 0x00FF00FF) << 8);  x = ((x & 0xFFFF0000) >> 16) | ((x & 0x0000FFFF) << 16); | 0x1 -> 0x80000000 |