二进制中1的位数及2的整数次幂

# 二进制中1的位数

## 背景知识

### 右移>>

**正数右移高位补0；负数右移高位补1。**

规律：**正数**右移31或32位，肯定为0000…00；

**负数**右移31位或32位，肯定为1111…11。

**0右移任意位都是0；-1右移任意位都是-1。**

System.out.println("4>>1 = " + (4>>1));//正数右移，高位补0

System.out.println("0>>1 = " + (0>>1));//**0右移任意位都是0**

System.out.println("-2>>1)) = " + (-2>>1));//负数右移，高位补1

System.out.println("-1>>1)) = " + (-1>>1));//**-1右移任意位都是-1**

### 左移<<

**左移时不分正负，符号位左移位溢出**。

**如1左移31位就是Integer的最小值1000**…00B;Integer的最小值左移1位就是0。

System.out.println("(Integer.MIN\_VALUE<<1) = " + (Integer.MIN\_VALUE<<1));//0

System.out.println("(-1<<1) = " + (-1<<1));//-2

System.out.println("1<<31)) = " + (1<<31) );//1左移31位就是Integer.MIN\_VALUE

## 最基本解法：对参数本身移位

/\*\*

\* 基于正数的方法

\* 传入负数时，会陷入无限循环中

\*/

public int NumberOf1\_1(int n){

int num = 0;

while(n != 0){

if((n&1) != 0) num++;

**n = n>>1;**

}

return num;

}

/\*\*

\* 比较牵强的做法：**改变右移为左移**

\*/

public int NumberOf1\_1\_1(int n){

int num = 0;

while(n != 0){

if((n&Integer.MIN\_VALUE) != 0) num++;

n = n<<1;

}

return num;

}

## 移动flag

/\*\*

\* 思路分析：由于n有正负之分，正数右移，左边补0；但是负数右移，左边补1，会陷入无限循环之中。

\* 变换思路：移动flag，由1逐渐左移，需要移动32次，即可判断比特1的位数。

\* @param n

\* @return

\*/

public int NumberOf1\_2(int n) {

**int flag = 1;**

int num = 0;

while(flag != 0){

if((n & flag) != 0) num++;

**flag = flag<<1;**

}

return num;

}

## 基于减1后与算法

/\*\*

\* 基于减1再与的方法

\* 原理：n减1，会导致原为1的最低位为0，且该位左边的所有位都变成1，再与原n做与操作，就消去了为1的1位；

\* 直到n变成0为止。

\* 如1101，不为0，num++; 减1变成1100,相与之后为1100，不为0，num++;再减1为1011，相与之后为1000，不为0，num++；

\* 再减1为0111，相与之后为0000.为0跳出循环。

\*/

public int NumberOf1\_3(int n){

int num = 0;

**while(n != 0){**

**num++;**

**n = (n-1)&n;**

**}**

return num;

}

# 判断是不是2的整数次幂

## 题目要求

利用**一条语句**判断一个正整数是不是2的整数次方。

## 思路分析

满足2的整数次方需要只有一个比特位为1。

前几种方法不能利用一个语句判断是否只有1个比特1；而第三种方法：基于减1后与的算法可以。

## Java代码：((n-1)&n)==0

**/\*\***

**\* 判断是不是2的幂**

**\*/**

**public boolean is2Power(int n){**

**if(n < 0) return false;**

**return ((n-1)&n) == 0;//n=0,也恰好返回true**

**}**

# 任意给定一个数，返回大于等于它的最小的2的乘方。

/\*\*

\* 题目介绍

\* 任意给定一个数，返回大于等于它的最小的2的整数次幂。

\*

\* 思路分析：

\* 主要思想：将该数最高位为1及之下的位全部设为1，然后加1返回即可。

\* 第一部分：保证所求最小的2的整数次幂中比特是1的那个比特位的紧挨着后面那个比特必须是1,且是第一个为1的比特。

\* n = n-1 的目的主要是针对n恰好是2的乘方。如0100,0000，需要减1，变成0011,1111;

\* 若n不是2的整数次幂，减1或减1都无所谓的，如0100,1000;减1变成0100,0111;

\* n= n-1的目的是保证所求最小的2的整数次幂中比特是1的那个比特位的紧挨着后面那个比特必须是1,且是第一个为1的比特。

\* 如0100,0000必须保证第3比特为1且是为1的第一个比特，如0011,0000;

\* 0100,1000必须保证第2比特为1且是为1的第一个比特，如0100,0111;

\* 第二部分：

\* 前提：第一部分已经保证了所求结果的为1的比特位的后面一位已经为1了。

\* 那么，右移1位然后再与，保证连续两位为1；

\* 再右移2位后再与，保证连续四位为1；

\* 再右移4位后再与，保证连续八位为1；

\* 再右移8位后再与，保证连续16位为1；

\* 再右移16位后再与，保证连续32位为1；

\* 最终肯定是保证了所需结果的为1的比特位的后面所有位都是1.

\* 之前的理解:

\* 若原数已经是2的幂，则只有一位为1，减1之后，得到后面的位全是1，下面不断右移再或，最后也是全是1，加1后恢复原值；

\* 若原数值不是2的幂，则减1之后，最高位1不变，然后先右移一位之后与原值或操作，则保证高两位全为1，

\* 再右移2位后与原值或操作，可以得到高四为全为1，紧接着右移4位再或，得到高8位为1，依次类推，

\* 最终可以所有数的后面位全为1。当然大多数不需要移动这么多位，为了保证全面性。

\*/

public int getMin2Power(int n){

n = n - 1;//保证结果的位1的比特位的后一位必须为1

n |= n >>> 1;

n |= n >>> 2;

n |= n >>> 4;

n |= n >>> 8;

n |= n >>> 16;

return (n < 0)?1:n+1;

}

# 整数m和n的位改变

## 题目介绍

输入两个整数m和n，计算需要改变m的二进制表中的多少位才能得到n。

示例：1010变成1101需要改变1101的3位才可以变成1010。

## 思路分析

需要两步:①首先异或m和n；②计算异或结果中1的位数。

## Java代码

/\*\*\*\*\*##m和n改变二进制位数##\*\*\*\*\*/

/\*\*

\* 输入两个整数m和n，计算需要改变m的二进制表中的多少位才能得到n。

\* 示例：1010变成1101需要改变1101的3位才可以变成1010。

\*/

public int bitNums(int m,int n){

int mn = m^n;

int num = 0;

while(mn != 0){

num++;

mn = (mn-1)&mn;

}

return num;

}

测试

@Test

public void test3(){

int n = 10,m=13;

int num = bitNums(m,n);

System.out.println(num);//4